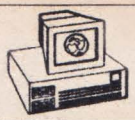
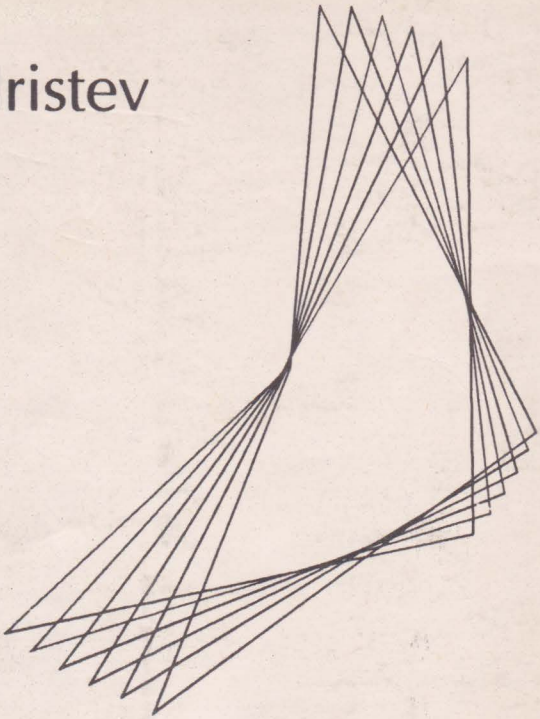


SERIA INFORMATICA

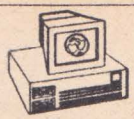
R. M. Hristev



GHIDUL UTILIZATORULUI SI PROGRAMATORULUI SPECTRUM

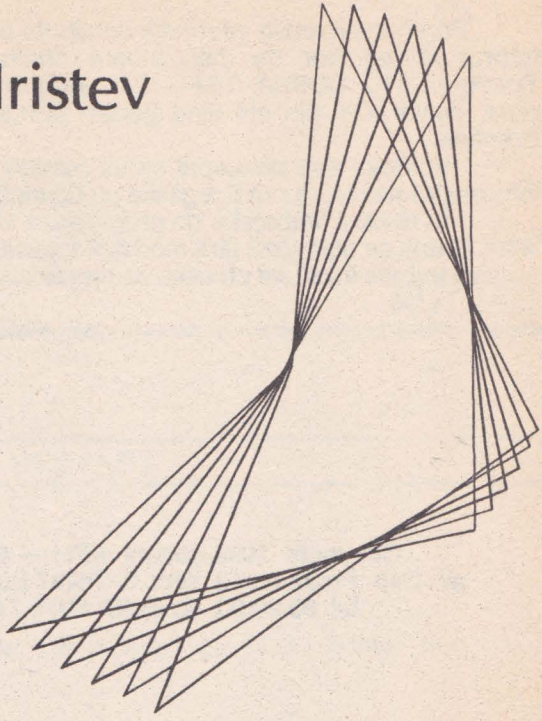
EDITURA APH

BUCURESTI 1992



SERIA INFORMATICĂ

R. M. Hristev



GHIDUL UTILIZATORULUI SI PROGRAMATORULUI SPECTRUM

EDITURA APH Bucureşti 1992

Prin numeroasele informaţii conţinute cartea se adresează tuturor utilizatorilor de calculatoare compatibile Sinclair ZX Spectrum (HC, COBRA, TIM-S, CIP, JET) conţinînd numeroase date, peste 90% din ele fiind pentru prima oara publicate în România

Volumul este conceput ca un manual de utilizare în care informaţiile necesare pot fi regăsite cu uşurintă.

În sintaxa limbajelor de programare simbolurile scrise cu litere **aldine** se utilizează fără modificări; cuvintele scrise cu litere *cursive* trebuie înlocuite cu ceea ce reprezintă ele

AUTORUL

Copyright 1992 Editura APH — SRL
str. Cap. Preda nr. 12, sect. 5, 76437 Bucureşti 69,
tel. 80.20.30, 80.93.97, 8074.77

Redactor, tehnoredactor, coperta: R. M. Hristev

Bun de tipar 20.8.1992. Apărut 1992. Format 70x100/16
Coli de tipar 11.5

Tiparul executat la Tipografia I. I. R. U. C.

1 RUTINELE ROM

1.1 RUTINELE RESTART

0000 Start
Rutină de inițializare, DE încărcat cu FFFF, adresa maximă posibilă pentru memoria RAM

0008 Eroare
Rutină de tratare a erorilor, RST 08 este urmat de DEFB n unde n=codul-erorii—1. Codul erorii: 0 pentru OK, ..., 1B pentru Tape loading error.

0010 Tipărire/Afișare caracter
Rutina de tipărire/afișare a unui caracter la canalul adresat de CURCHL.
La intrare: în A codul caracterului, celelalte informații în variabilele de sistem.

0018 Culege caracter
Conținutul locației adresate de CH-ADD este adus în HL, return dacă caracterul este afișabil, dacă nu, CH-ADD este incrementat și se repetă testul.

0020 Culege următorul caracter
Rutină utilizată la parcurgerea liniilor de program BASIC de către interpretor.

0028 Calcul în virgulă flotantă
Rutina de calcul în virgulă flotantă, fiecare apel RST 28 este urmat de o serie de DEFB care se termină cu DEFB 38, fiecare byte definind un cod de operație.

0030 BC spații
Rutina creează locații libere în zona de lucru, utilizează WORKSP care conține adresa de start a spațiului de lucru.
La intrare: BC conține numărul locațiilor

0038 Întrerupere mascabilă
Rutină de tratare a întreruperilor mascabile,se incrementează ceasul de timp real FRAMES la fiecare 20ms si se citește tastatura..

0053 Continuare RST 08

Adresa de întoarcere a interpretorului este o locaţie DEFB care conţine codul erorii întîlnite. DEFB este transferat în ERR-NR.. Stiva este ştearsă. Salt pentru resetarea stivei-calculator.

0066 Întrerupere nemascabilă

Rutină de tratare a întreruperilor nemascabile; neutilizată în Spectrum original, aceasta conţine o eroare: instrucţiunea JR NZ trebuie înlocuită cu JR Z (utilizează NMIADD)

0074 CH-ADD+1

Rutina incrementează CH-ADD; încarcă A cu locaţia adresată de CH-ADD+1

007D Sare peste

Rutina testează dacă codul din A este afişabil; setează HL şi CH-ADD în concordanţă cu A.

0095 Tabela cuvintelor cheie

Tabela cuvintelor-cheie BASIC după care se face afişarea lor; cuprinde: ?, RND, ... RETURN, COPY. Bitul 7 al ultimei litere al fiecărui cuvînt este setat.

0205 Tabela tastelor (1)

codurile simbolurilor obţinute prin: cursor L şi CS; cuprinde: B, H, ...Q, A.

022C Tabela tastelor (2)

codurile simbolurilor obţinute prin: cursor E, taste literare; cuprinde: READ, BIN, ... STR$, LN.

0246 Tabela tastelor (3)

codurile simbolurilor obţinute prin: cursor E, taste literare şi SS; cuprinde: ~, BRIGHT , ..., [, BEEP.

0260 Tabela tastelor (4)

codurile de control obţinute prin: taste numerice şi CS; cuprinde: DELETE, EDIT, ..., cursor-dreapta, GRAPHICS.

026A Tabela tastelor (5)

codurile simbolurilor obţinute prin: taste literare şi SS; cuprinde: STOP, *, ..., AND, :.

0284 Tabela tastelor (6)

codurile simbolurilor obţinute prin: cursor E, taste numerice şi SS; cuprinde: FORMAT, DEF FN, ..., POINT, CAT.

1.2 RUTINELE DE TASTATURĂ

028E Scanarea tastaturii

Rutina principală de citire a tastaturii de 40 de taste.

La ieşire: în E valoare de la 00 la 27, codul tastei sau FF, nici o tastă apasată
în D codul tastei shift apăsate sau FF (în DE este FFFF dacă nici o tastă nu a fost apăsată)

Portul tastaturii (în C): FE

02BF Tastatura

Rutina de citire a tastaturii apelată la fiecare 20ms, decodifică tasta apasată, pune codul în LAST-K şi bitul 5 al FLAGS este setat. Se utilizează 2 sisteme de variabile KSTATE0 ... KSTATE3 şi KSTATE4 ... KSTATE7. Un set devine disponibil pentru o nouă tastă numai după 1/10s adică 5 apelări ale rutinei.

0308 Repetarea tastei

Este o subrutină a **Tastatura** (02BF); o tastă se consideră repetată prima oară după o perioadă REPDEL (normal 0.7s) şi ulterior după o perioadă REPPER (normal 0.1s).

031E K-test

Rutina testează dacă nu a fost apăsată nici o tastă sau numai o tastă shift: dacă da return, dacă nu calculează codul tastei.

0333 Decodarea tastei

La intrare: în E codul tastei
în D valoarea FLAGS
în C modul cursor

La ieşire: în A codul simbolului , pentru tasta sau ansamblul de taste apăsate

1.3 RUTINE DIFUZOR

03B5 Beeper

Rutina care activează şi dezactivează difuzorul, repetat, astfel încît se obţine sunetul.

La intrare: în DE produsul dintre frecventa sunetului si durata lui (in s)
în HL contor pentru bucla de timp=3.5*106/8/f-30.125

03F8 BEEP

La intrare: pe stiva-calculator, în vîrf, se găseşte "înălţimea" notei şi sub ea "durata" notei

Rutina efectuează verificări asupra "înălţimii" şi "duratei" sunetului şi pregăteşte parametrii pentru BEEPER pe care o apelează.

046E Tabela frecvenţelor

261.63	do	369.99	fa#
277.18	do#	392	sol
293.66	re	415.30	sol#
311.13	re#	440	la
329.63	mi	466.16	la#
349.23	fa	493.88	si

Observaţie: La fiecare salt la octava superioară respectiv inferioară frecvenţele se dublează, respectiv înjumătăţesc.

04AA Neutilizat

1.4 RUTINELE CASETOFON

Pentru toate rutinele:
DE conţine lungimea blocului de bytes
IX conţine adresa de început a blocului de bytes
A conţine 00 pentru header, FF pentru program/date

04C2 SA-bytes
Rutină de salvare a blocurilor de bytes.

053F SA/LD-Ret
Rutină de retur după o operaţie de salvare sau încărcare.

0556 LD-bytes
Rutină de citire/verificare a blocurilor de bytes.

05E3 LD-edge
La intrare: în B constanta de timp
în C tipul de margine
La ieşire: în F: C=0 la eroare, Z=0 la Break

0605 SAVE, LOAD, VERIFY, MERGE
Rutina este comună comenzilor SAVE, LOAD, VERYFY şi MERGE.
TADDR este E0 pentru SAVE E1 pentru LOAD
E2 pentru VERIFY E3 pentru MERGE
Header-ul este format în zona de lucru, astfel:
locaţia: IX+00 conţine tipul
IX+01 pînă la IX+0A conţine numele (FF în IX+01 dacă este nul)
IX+0B şi IX+0C conţin lungimea în bytes
IX+0D pînă la IX+10 conţin informaţii dependente de tipul blocului

07CB VERIFY
Rutina VERIFY, datele sînt numai comparate nu încărcate.

0802 LOAD date
Rutină comuna pentru LOAD şi VERIFY.

0808 LOAD
Rutină de încărcare a unui program BASIC şi a datelor sale.

08B6 MERGE
Rutina efectuează: a)încărcărea unui bloc de bytes
b)suprapune liniile BASIC
c)suprapune variabilele programelor BASIC

092C MERGE linie sau variabilă
La intrare: în F:. C=0 suprapunerea unei linii BASIC
C=1 suprapunerea unei variabile
Z=0 adăugare
în HL adresa începutului noii intrări
în DE adresa unde se face înlocuirea

0970 SAVE
Rutină de salvare a programelor.

09A1 Mesaje-bandă magnetică
Bitul 7 al ultimului caracter este setat.

```
09A1 DEFB    80
09A2 DEFM    "Start tape then press any key"
09C1 DEFM    carriage return,"Program:"
09CB DEFM    carriage return, "Number array:"
09DA DEFM    carriage return, "Character array:"
09EC DEFM    carriage return, "Bytes:"
```

1.5 RUTINE ECRAN

09F4 Print-out

Rutina print principală.

La intrare: în A: codul caracterului (afişabil sau de control)
în variabilele de sistem: parametrii de poziţie (rutina 0B03)

0A11 Tabela caracterelor de control

4E	PRINT comma	4F	neutilizat
57	Edit	5F	INK control
10	cursor stînga	5E	PAPER control
29	cursor dreapta	5D	FLASH control
54	cursor jos	5C	BRIGHT control
53	cursor sus	5B	INVERSE control
52	DELETE	5A	OVER control
37	Enter	54	AT control
50	neutilizat	53	TAB control

0A23 Cursor stînga

La intrare: în B linia curentă
în C coloana curentă

Observaţie: instrucţiunea LD A, 18 trebuie înlocuită cu LD A, 19.

0A3D Cursor dreapta

Analog cursor stînga (rutina 0A23).

Observaţie: returul ar trebui efectuat la adresa 0ADC.

0A4F Carriage return

Se efectuează test **scroll?** înainte de decrementarea numărului de linie.

0A5F Print comma

A este încărcat cu 00h pentru TAB 0 sau 10h pentru TAB 16.

0A69 Afişează ?

Dacă se încearcă afişarea unui caracter neafişabil atunci se afişează **?**.

0A6D Caracter de control al operanzilor

INK ... OVER au un unic operand; AT şi TAB au 2 operanzi

Rutina încarcă caracterul de control în TVDATA-inf, primul operand în TVDATA-sup sau în A dacă este operand unic şi cel de al 2-lea operand în A.

0AD9 Afişează caracter

Un caracter este afişat prin apelarea rutinei 0B24 urmată de rutina 0ADC.

0ADC Memorează poziţia

Noua poziţie linie, coloană şi adresa pixel sînt încărcate în variabilele de sistem

0B03 Aduce poziţia

Poziţia curentă este adusă din sistemul de variabile pentru tipărire/afişare

0B24 Afişează un caracter

Rutină folosită pentru afişarea caracterelor: matrice 8*8 pixeli
La intrare: în DE adresa de bază a formei caracterelor (normal 3D00)
în HL adresa destinaţie
în BC linia şi coloana curentă

0BDB Actualizează atributul

Rutina actualizează atributul de culoare al caracterului, utilizind valoarea anterioară şi ATTR-T, MASK-T, P-FLAG.

0C0A Afişează mesaj

Rutina afişează cuvinte-cheie BASIC sau mesaje.

0C3B Afişare repetată

Rutină utilizată la afişarea repetată a unui caracter

0C55 Test scroll

Rutina testează necesitatea unui scroll în cazurile:
a)la întîlnirea unui carriage return
b)AT în INPUT
c)linia curentă plină
La intrare: în B: numărul liniei pentru care se doreşte testul
Observaţie: La 0CF8 mesajul scroll? ,cu marker iniţial şi bitul 7 al ultimului caracter setat.

0D4D Copierea temporară a atributelor de culoare

Rutină utilizată cînd atributele permanente trebuie copiate în variabilele de sistem.

0E0D Comanda CLS

Rutina şterge tot ecranul: resetează pixelii şi setează atributele conform ATTR-P apoi prelucrează partea de jos a ecranului (normal liniile 22 şi 23).

0DAF Şterge ecranul

Această rutină este utilizată de:
a)comanda CLS
b)rutina principală de execuţie
c)rutina de listare

0DD9 Clear set

La intrare: în BC linia şi coloana ecran
sau în C coloana din tamponul de imprimantă
Se calculează adresa primului bit al caracterului.

0DFE Scroll

Rutina efectuează un scroll pentru numărul de linii din registrul B.

0E44 Şterge linii

Rutina şterge numărul de linii din registrul B, din partea de jos a ecranului

0E88 Atribute

Rutina efectuează:
a)pentru o adresă din ecran adresa atributului este returnată în HL (adresa de intrare: cea dea 9-a linie a caracterului)
b)pentru o linie dată în B este returnată adresa de start a liniei în BC

0E9B Adrese

Rutina calculează, pentru o linie dată în B, adresa din ecran.

0EAC Copy

Rutina copiază rînd cu rînd ecranul la imprimantă (176 rinduri)

0EDC Copy buffer

Rutina este apelată la fiecare golire a tamponului de imprimantă .

0EDF Şterge tamponul de imprimantă

0EF4 Copiază linie la imprimantă

La intrare: în HL adresa de bază a primului rînd al liniei
în B numărul rîndului

0F2C Editor

Rutina este utilizată de:

a)rutina principală la intrarea unei noi linii BASIC
b)comanda INPUT

0F81 Adaugă un caracter

Rutina adaugă codul unui caracter în linie EDIT sau INPUT.

0FA0 Tabela tastelor editor

09	Edit	70	Delete
66	cursor stînga	7E	Enter
6A	cursor dreapta	CF	SS
50	cursor jos	D4	Graphics
B5	cursor sus		

0FA9 Tasta Edit

Rutina este utilizată în:

a)modul Edit: prin apăsarea tastei Edit este pregatită o linie pentru editare
b)INPUT: şterge răspunsul curent şi alocă spaţiu pentru unul nou

1007 Editare-cursor stînga

Rutina se termină în Editare-cursor dreapta (rutina 100C)

100C Editare-cursor dreapta

1015 Editare-delete

Cursorul se mişcă în stînga

101E Editare-ignoră 2 coduri

1024 Editare-Enter

Variabilele de sistem, de tratare a erorilor, sînt actualizate şi salt la rutinele de tratare ale erorilor dacă este cazul.

1031 Editare margine

Adresa cursorului din HL este decrementată dacă cursorul nu se află la marginea liniilor dar astfel încît cursorul nu este pus între caracterele de control şi parametrii lor.

1059 Editare-cursor sus

1076 Editare-SS şi Graphics

Rutina tratează utilizarea codurilor SS şi Graphics.

107F Editare-erori

Rutină de tratare a erorilor la editare.

1097 Şterge zona de lucru

Rutina şterge zona de editare sau spaţiul de lucru.

10A8 Input de la tastatură

Rutina returnează codul ultimei taste apăsate.

111D Copiere ecran-jos
Rutina este apelată pentru afişarea unei linii din zona de editare sau în cazul unui INPUT în partea de jos a ecranului.
1190 Set HL şi DE
La ieşire: în HL adresa primei locaţii
în DE adresa ultimei locaţii
din zona de editare sau spaţiul de lucru.
11A7 Retrage-virgulă flotantă
Rutina plasează numerele în virgulă flotantă, din linia BASIC în curs de interpretare pe stiva-calculator.

1.6 RUTINE DE EXECUŢIE

11B7 Comanda NEW
11CB Iniţializare
Rutina de start sau NEW, iniţializează variabilele de sistem şi afişează mesajul "@1982 Sinclair Research Ltd."
12A2 Rutina principală de execuţie
Rutina controlează editarea, execuţia comenzilor directe şi producerea mesajelor de eroare.
1391 Tabela mesajelor de eroare

1391	DEFB 80	marker iniţial
1392	Eroare-0	OK
1394	Eroare-1	NEXT without FOR
...		
1516	Eroare-Q	Parameter error
1539	", "	(virgulă şi blank)
1539	"@ 1982 Sinclair Research Ltd."	

1555 Eroare-G
155D Adaugă o linie BASIC
Rutina introduce o linie nouă de program în zona programului BASIC. Noua linie o înlocuieşte pe cea veche dacă este cazul. O linie program care conţine doar numărul de linie nu este introdusă.
15AF Tabela de informaţii pentru canale
Există 4 canale iniţiale de comunicare:
K pentru tastatură — R pentru spaţiul de lucru
S pentru ecran — P pentru imprimantă
Pentru fiecare canal: adresa rutinei out, adresa rutinei in.

15AF	DEFB F4, 09	rutina 09F4
	DEFB AB, 10	rutina 10AB
	DEFB 4B	canal K
15B4	DEFB F4, 09	rutina 09F4
	DEFB C4, 15	rutina 15C4

```
        DEFB 5          canal S
15B9    DEFB 81, 0F     rutina 0F81
        DEFB C4,15      rutina 15C4
        DEFB 52         canal R
15BE    DEFB F4, 09     rutina 09F4
        DEFB C4,15      rutina 15C4
        DEFB 50         canal P
15C3    DEFB 80         marcator de sfîrşit
```

15C4 Eroare-J

15C6 Tabela de informaţii stream

```
15C6 DEFB 01, 00    stream FD   canal K
15C8 DEFB 06, 00    stream FE   canal S
15CA DEFB 0B, 00    stream FF   canal R
15CC DEFB 01, 00    stream 00   canal K
15CE DEFB 01, 00    stream 01   canal K
15D0 DEFB 06, 00    stream 02   canal S
15D2 DEFB 10, 00    stream 03   canal P
```

15D4 Asteaptă o tastă

Rutină de control pentru rutina curentă de input.

15E6 Adresa input

Rutina salvează regiştrii şi introduce în HL adresa de input.

15EF Rutina principală de afişare

La intrare: în A codul caracterului

1601 Deschide canal

La intrare: în A număr de stream valid (normal între FD şi 03); în funcţie de stream va fi deschis canalul respectiv

1615 Flags canale

Rutina setează variabilele de sistem conform canalelor utilizate.

162D Tabela adrese canale

```
162D    DEFB 4B,06      canal K     offset 06       adresa 1634
162F    DEFB 53,12      canal S     offset 12       adresa 1642
1631    DEFB 50,1B      canal P     offset 1B       adresa 164D
1633    DEFB 00         marcator de sfîrşit
```

1634 Flags canal K

1642 Flags canal S

164D Flags canal P

1652 Deschide fereastră

Rutina deschide o fereastră în memorie.

La intrare: în HL adresa locaţiei de după fereastră
în BC lungimea ferestrei

Atenţie: dacă se doreşte eliberarea unei singure locaţii, atunci adresa de început a rutinei este 1652, dacă nu atunci 1655

La ieşire: în HL adresa locaţiei dinaintea ferestrei
în DE adresa ultimei locaţii din fereastră

Observaţie: noile locaţii păstrează vechile valori, fereastra este de la HL+1 pîna la DE, dar în unele rutine se consideră de la HL+2 la DE+1.

1664 Pointeri

Rutina actualizează variabilele de sistem, dacă este cazul, cînd o fereastră de memorie este deschisă sau modificată..

La intrare: în BC numărul locaţiilor
în HL adresa locaţiei dinaintea poziţiei

168F Aduce număr de linie

La intrare: în HL adresă locaţie
La ieşire: în DE numărul liniei care începe la HL
Dacă HL nu reprezintă adresa primului byte a numărului de linie atunci la ieşire DE=0 (168F DEFB 00, 00 număr de linie 0)

169E Rezervă

Rutină utilizată normal de RST 30, fereastra este creată între zona de lucru şi stiva-calculator.

16B0 Set minim

Rutina resetează zona de editare, spaţiul de lucru şi stiva-calculator la mărimile minime.

16D4 Apelează editorul de linie

16DB Indexator

Rutină folosită pentru parcurgerea tabelelor.

16E5 Comanda CLOSE#

Rutină pentru închiderea stream-urilor (00 la 03 nu pot fi închise)

1716 Tabela de adrese close stream

1716	DEFB 4B,05	canal K	offset 05	adresa 171C
1718	DEFB 53,03	canal S	offset 03	adresa 171C
171A	DEFB 50,01	canal P	offset 01	adresa 171C

Observaţie: nu există marcator de sfîrşit

171C Închide stream

171E Date stream

La intrare: pe stiva-calculator se află stream-ul
La ieşire: în BC date stream

1736 Comanda OPEN#

Rutină pentru deschiderea stream-urilor. Canale valide K, k,S, s, P, p.

175D Subrutina OPEN#

Subrutină a rutinei 1736.

177A Tabela de adrese de deschidere stream

177A	DEFB 4B,06	canal K	offset 06	adresa 1781
177C	DEFB 53,08	canal S	offset 08	adresa 1785
1780	DEFB 50,0A	canal P	offset 0A	adresa 1789
1780	DEFB 00	marcator de sfîrşit		

1781 Deschide K

1785 Deschide S

1789 Deschide P

1793 Comenzi CAT, ERASE, FORMAT, MOVE

Rutinele acestor comenzi se află în ZX Interface 1 astfel încît utilizarea acestora fără interfaţa respectivă generează eroare O.

1795 Comenzile LIST şi LLIST

17F5 Punct de intrare LLIST

17F9 Punct de intrare LIST

1855 Afişează o linie BASIC

La intrare: în HL adresa primei locaţii a liniei

18B6 Number
Dacă A=0E atunci HL este incrementat cu 6 (sare peste un număr în virgulă flotantă).
18C1 Afişează un caracter tip FLASH
Rutină apelată pentru afişarea cursorului şi cursorului de eroare.
18E1 Afişează cursorul
Rutina afişează cursorul K, L,C, E sau G.
190F Culege linie BASIC
La intrare: în HL adresa S-TOP sau E-PPC şi le modifică cu numărul liniei următoare
1925 Afişează linie BASIC
Rutina afişează caracterele sau cuvintele-cheie dintr-o linie BASIC.
196E Adresa liniei BASIC
La intrare: în HL numărul liniei
La ieşire: în HL adresa liniei sau a primei linii de după numărul dat
în DE adresa liniei anterioare
1980 Compară numere de linie BASIC
La intrare: în BC număr de linie BASIC
în HL adresa unei linii BASIC
La ieşire: în F: C=1 pentru egalitate
1988 Neutilizat
198B Caută toate instrucţiunile
La intrare: în D numărul instrucţiunii dintr-o linie BASIC
sau în E codul unui cuvînt-cheie
în HL adresa locaţiei de unde începe căutarea
La ieşire: în HL adresa locaţiei dinaintea instrucţiunii
19B8 Următoarea linie/variabilă
Rutina caută urmăroarea linie BASIC sau următoarea variabilă BASIC (există 6 tipuri)
La intrare: în HL adresa curentă
La ieşire: în HL adresa următoare
în DE adresa curentă
19DD Diferenţa
Rutina calculează diferenţa cu transport între HL şi DE, returnează rezultatul în BC apoi reface HL şi DE şi le inversează.
19E5 Recuperează
Rutina mută blocuri de bytes.
Punct de intrare 19E5: La intrare: în DE adresa sursă
în HL adresa destinaţie
Punct de intrare 19E8: La intrare: în DE adresa sursă
în HL adresa destinaţie
în BC numărul de bytes
19FB Editare-număr linie
La ieşire: în BC numărul liniei din zona editare, dacă nu există atunci BC=0 (comandă directă)
1A1B Afişare număr linie
Punct de intrare 1A1B: afişează numărul de linie din BC
Punct de intrare 1A28: afişează numărul de linie de la adresa HL

1.7 INTERPRETORUL BASIC

1A48 Tabela offset pentru comenzile BASIC

	DEFB	adresă	comandă		DEFB	adresă	comandă
1A48	B1	1AF9	DEF FN	1A61	94	1AF5	BORDER
1A49	CB	1B14	CAT	1A62	56	1AB8	CONTINUE
1A4A	BC	1B06	FORMAT	1A63	3F	1AA2	DIM
1A4B	BF	1B0A	MOVE	1A64	41	1AA5	REM
1A4C	C4	1B10	ERASE	1A65	2B	1A90	FOR
1A4D	AF	1AFC	OPEN#	1A66	17	1A7D	GO TO
1A4E	B4	1B02	CLOSE#	1A67	1F	1A86	GO SUB
1A4F	93	1AE2	MERGE	1A68	37	1A9F	INPUT
1A50	91	1AE1	VERIFY	1A69	77	1AE0	LOAD
1A51	92	1AE3	BEEP	1A6A	44	1AAE	LIST
1A52	95	1AE7	CIRCLE	1A6B	0F	1A7A	LET
1A53	98	1AEB	INK	1A6C	59	1AC5	PAUSE
1A54	98	1AEC	PAPER	1A6D	2B	1A98	NEXT
1A55	98	1AED	FLASH	1A6E	43	1AB1	POKE
1A56	98	1AEE	BRIGHT	1A6F	2D	1A9C	PRINT
1A57	98	1AEF	INVERSE	1A70	51	1AC1	PLOT
1A58	98	1AF0	OVER	1A71	3A	1AAB	RUN
1A59	98	1AF1	OUT	1A72	6D	1ADF	SAVE
1A5A	7F	1AD9	LPRINT	1A73	42	1AB5	RANDOMIZE
1A5B	81	1ADC	LLIST	1A74	0D	1A81	IF
1A5C	2E	1A8A	STOP	1A75	49	1ABE	CLS
1A5D	6C	1AC9	READ	1A76	5C	1AD2	DRAW
1A5E	6E	1ACC	DATA	1A77	44	1ABB	CLEAR
1A5F	70	1ACF	RESTORE	1A78	15	1A8D	RETURN
1A60	48	1AA8	NEW	1A79	5D	1AD6	COPY

Adresele generate sînt adresele din tabela de parametri pentru comenzile respective.

1A7A Tabela de parametri

Reprezintă tabela parametrilor comenzilor BASIC de la LET (1A7A) pîna la CAT (1B14), adresele fiind date de tabela 1A48. Pentru fiecare comandă există 1 pînă la 8 bytes, offset de sintaxă, care se citesc în modul următor:

Dacă byte-ul este între 00 şi 0B atunci (byte de sintaxă):

- 00 nu mai urmează nici un operand
- 01 utilizat în LET, urmează o variabilă
- 02 utilizat în LET, urmează o expresie numerică
- 03 urmează o expresie numerică, în lipsă se consideră 0
- 04 urmează o variabilă al cărei nume este un singur caracter
- 05 urmează o serie de articole
- 06 urmează o expresie numerică
- 07 urmează coduri de culoare
- 08 urmează 2 expresii numerice separate de virgulă
- 09 analog 08, expresiile pot fi precedate de coduri de culoare
- 0A urmează o expresie şir
- 0B pentru rutinele de bandă magnetică

byte-ul 00, 03 sau 05 este urmat de o adresă a comenzii de executat. Ceilalţi bytes reprezintă coduri caracter sau cuvînt-cheie.

1B17 Interpretorul

Rutina verifică sintaxa liniei BASIC.

1B28 Bucla de comenzi

Rutina execută comenzile dintr-o linie BASIC.

1B6F Separator

Dacă separatorii necesari nu sînt găsiţi se generează eroare C.

1B76 Întoarcere după o comandă

După execuţia unei comenzi se efectuează o întoarcere în acest punct.

1B8A Run

Punct de intrare pentru rularea unei linii din zona de editare sau pentru verificarea sintaxei dintr-o linie din zona de editare care are mai multe comenzi.

1B9E Linie BASIC nouă (1)

La un salt în program, rutina calculează adresa noii linii.

1BB2 Comanda REM

1BB3 Sfîrşit de linie BASIC

La verificarea sintaxei: return
La rulare: actualizează NXTLIN

1BBF Linie BASIC nouă (2)

La ieşire: în PPC numărul noii linii BASIC
în HL adresa

1BD1 Linie BASIC nouă (3)

La intrare: în HL adresa locaţiei după sfîrşitul liniei noi
în DE adresa locaţiei dinaintea primului caracter al liniei

Se actualizează NXTLIN.

1BEE Caută sfîrşitul liniei BASIC

1BF4 Comanda următoare

Dacă Enter atunci comanda următoare este pe linia următoare.
Dacă : atunci comanda următoare este pe aceeaşi linie, în continuare.
Altfel eroare.

1C01 Tabela offset pentru bytes de sintaxă

	DEFB	adresă	offset		DEFB	adresă	offset
1C01	0F	1C10	00	1C07	7B	1C82	06
1C02	1D	1C1F	01	1C08	8E	1C96	07
1C03	4B	1C4E	02	1C09	71	1C7A	08
1C04	09	1C0D	03	1C0A	B4	1CBE	09
1C05	67	1C6C	04	1C0B	81	1C8C	0A
1C06	0B	1C11	05	1C0C	CF	1CD8	0B

1C0D Sintaxă offset 03

1C10 Sintaxă offset 00

1C11 Sintaxă offset 05

1C16 Salt la rutina de execuţie a comenzii

1C1F Sintaxă offset 01

1C22 Asignarea variabilelor DEST şi STRLEN

1C4E Sintaxă offset 02

1C56 Asignează o valoare

1C6C Sintaxă offset 04
1C79 Evaluarea epresiei numerice/şir
Rutină de evaluare a unei expresii numerice sau şir
1C7A Sintaxă offset 08
1C82 Sintaxă offset 06
1CBC Sintaxă offset 0A
1C96 Sintaxă offset 07
1CBE Sintaxă offset 09
Rutină utilizată de PLOT, DRAW şi CIRCLE.
1CDB Sintaxă offset 0B
1CDE Culege un număr
Rutina evaluează o expresie numerică; dacă nu există returnează 0.
1CEE Comanda STOP
1CF0 Comanda IF
1D03 Comanda FOR
1D86 Caută în program
Rutina caută instrucţiuni DATA, DEF FN sau NEXT în program (cele care vor fi utilizate).
La intrare: în E codul cuvîntului-cheie
în HL adresa de început a căutării
1E27 Comanda DATA
1E42 Comanda RESTORE
1E4F Comanda RANDOMIZE
La intrare: în BC operandul, dacă BC=0 se utilizează FRAMES
La ieşire: în SEED, BC sau FRAMES
1E5F Comanda CONTINUE
1E67 Comanda GO TO
1E7A Comanda OUT
1E80 Comanda POKE
1E86 2 parametri
Rutină utilizată de POKE.
Numărul din vîrful stivei-calculator este comprimat în A, în complement faţă de 2.
Numărul al 2-lea din stiva-calculator este comprimat în BC.
1E94 Găseşte întregul (1)
Numărul din vîrful stivei-calculator este comprimat în A.
1E99 Găseşte întregul (2)
Numărul din vîrful stivei-calculator este comprimat în BC.
1EA1 Comanda RUN
1EAC Comanda CLEAR
1EED Comanda GO SUB
1F05 Test fereastră memorie
Testează dacă există spaţiu suficient în memorie pentru următoarea operaţie.
1F1A Memorie liberă
Rutina calculează cît spaţiu disponibil există în memorie (echivalent cu PRINT 65536-USR 7962).
1F23 Comanda RETURN

1F3A Comanda PAUSE
1F54 Tasta BREAK
Rutină pentru citirea tastei Break
La ieşire: în F: C=0 numai dacă SS şi Break au fost apăsate
1F60 Comanda DEF FN
1FC3 Unstack-z
Rutină utilizată pentru întoarcere rapidă dintr-o rutină la verificarea sintaxei.
1FC9 Comenzile LPRINT şi PRINT
1FF5 Print carriage return
1FFC Caractere de control ale PRINT, PRINT şi INPUT
Subrutină a comenzilor PRINT, LPRINT şi INPUT care manipulează AT şi TAB.
2045 Sfîrşitul afişării
204E Poziţia de afişare
Rutina manipulează caracterele de control ale afişării (:, ').
2070 Utilizare stream
Rutină utilizată cînd se folosesc diverse stream-uri.
2089 Comanda INPUT
21B9 Asignare INPUT
Rutină utilizată la verificarea sintaxei şi la atribuire de valori variabilelor.
21D6 Utilizarea canalului K
La ieşire: în F: Z=1 la utilizarea tastaturii
21E1 Culorile de afişare (1)
Rutină de manipulare INK, PAPER şi OVER.
2211 Culorile de afişare (2)
Rutină de actualizare a sistemului de variabile.
226C Culorile de afişare (3)
Rutină de manipulare FLASH şi BRIGHT.
2294 Comanda BORDER
22AA Adresa pixel
Rutină utilizată de funcţia POINT şi comanda PLOT.
La intrare: în BC coordonatele pixel-ului
La ieşire: în HL adresa byte-ului în care se găseşte pixel-ul
în A poziţia pixel-ului din byte
22CB Funcţia POINT
La intrare: adresa pixel-ului pe stiva-calculator
La ieşire: în A: 0 dacă pixel-ul are culoare PAPER
1 dacă pixel-ul are culoare INK
22DC Comanda PLOT
La intrare: adresa pixel-ului pe stiva-calculator; acesta este afişat conform INVERSE şi OVER din P-FLAG
2307 Stiva-calculator în BC
Rutina încarcă în BC 2 numere în virgulă flotantă de pe stiva-calculator, acestea trebuie să fie în intervalul 00, FF.
2314 Stiva-calculator în A
Rutina încarcă în A un număr în virgulă flotantă de pe stiva-calculator, acesta trebuie sa fie în intervalul 00, FF.
2320 Comanda CIRCLE

La intrare: în virful stivei-calculator: R raza cercului
pe stivă, sub R, este coordonata Y
pe stivă, sub Y, este coordonata X

2382 Comanda DRAW

Dacă sînt daţi 2 parametri X şi Y se trasează un segment de dreaptă din poziţia X0, Y0 din COORDS pînă în poziţia X0+X, Y0+Y.
Dacă sînt daţi 3 parametri X, Y, G se trasează un arc de cerc, între aceleaşi puncte, de lungime G radiani, în sens trigonometric (G>0) sau orar (G<0).
Ordinea pe stiva-calculator, de la vîrf la bază: X, Y sau X, Y, G.

247D Parametrii iniţiali

Rutină utilizată de comenzile CIRCLE şi DRAW pentru pregătirea parametrilor iniţiali de trasare.

247B DRAW

Rutină utilizată de comanda DRAW (rutina 2382).
La intrare: X şi Y pe stiva-calculator
Rutina trasează efectiv linia între X0, Y0 şi X0+X, Y0+Y.

1.8 EVALUATORUL DE EXPRESII

24FB Scanare

Rutina evaluează expresia următoare.
La ieşire: un rezultat pe 5 bytes pe stiva-calculator
Dacă rezultatul este număr în virgulă flotantă atunci în FLAGS bit6=1
Dacă rezultatul este un şir atunci: primul byte: nespecificat
bytes 2,3: adresa şirului
bytes 4,5: lungimea şirului

2530 Sintaxa-Z

în FLAGS: bit7=0 înseamnă execuţie
bit7=1 înseamnă verificarea sintaxei

2535 Scanare SCREEN$

Rutină utilizată de funcţia SCREEN$.
La intrare: pe stiva-calculator coordonatele X, Y
La ieşire: pe stiva-calculator codul caracterului
Rutina utilizează tabela caracterelor, adresată de CHARS.

2596 Tabela funcţiilor de scanare

nume	adresă	cod	offset	adresa rutinei
"	2596	22	1C	25B3
( )	2598	28	4F	25E8
zecimal	259A	2E	F2	268D
număr, 1 caracter	259C	2B	12	25AF
FN	259E	A8	56	25F5
RND	25A0	A5	57	25F8
PI	25A2	A7	84	2627
INKEY$	25A4	A6	8F	2634

BIN	25A6	C4	E6	268D
SCREEN	25A8	AA	BF	2668
ATTR	25AA	AB	C7	2672
POINT	25AC	A9	CE	267D
-	25AE	00	marcator de sfirsit	

25AF Scanare număr, 1 caracter

La întîlnirea caracterului +, scanarea continuă cu caracterul următor.

25B3 Scanare "

25EB Scanare ( )

25F5 Scanare FN

25F8 Scanare RND

2627 Scanare PI

2634 Scanare INKEY$

2668 Scanare SCREEN$

2672 Scanare ATTR

267B Scanare POINT

2684 Scanare caracter alfanumeric

268D Scanare zecimal/BIN

26C9 Scanare variabile

Rutină pentru recunoaşterea variabilelor şi asignare de valori.

2795 Tabela operatorilor

	cod	operator	cod operator		cod	operator	cod operator
2795	2B	+	CF	27A3	3C	<	CD
2797	2D	-	C3	27A5	C7	<=	C9
2799	2A	*	C4	27A7	C8	>=	CA
279B	2F	/	C5	27A9	C9	<>	CB
279D	5E	^	C6	27AB	C5	OR	C7
279F	3D	=	CE	27AD	C6	AND	C8
27A1	3E	>	CC	27AF	00	marcator de sfirşit	

27B0 Tabela de priorităţi a operatorilor

	operator	prioritate		operator	prioritate
27B0	-	06	27B7	>=	05
27B1	*	08	27B8	<>	05
27B2	/	08	27B9	>	05
27B3	^	0A	27BA	<	05
27B4	OR	02	27BB	=	05
27B5	AND	03	27BC	+	06
27B6	<=	05			

27BD Scanare funcţii FN

28AB Trece peste funcţie

Rutina parcurge DEF FN fără a modifica CH-ADD.

28B2 Caută variabile

Rutină utilizată pentru căutarea variabilelor sau argumentelor dintr-o instrucţiune DEF FN.

La intrare: în CH-ADD adresa primei litere a varibilei căutate din program sau zona de lucru
DEFADD<>0 implică căutare de funcţie

La ieşire: dacă variabila a fost găsita, în F: C=0;

în F: C=0, Z=1 pentru şiruri şi tablouri,
în HL adresa ultimei/singurei litere a variabilei
dacă variabila nu a fost găsită
în F: C=1, Z=1 pentru tablouri
în HL adresa primei litere a variabilei din linia BASIC
în ambele cazuri în C bit 5, 6 indică tipul variabilei

2951 Caută argumente DEF FN

Subrutină a 28B2 dacă DEFADD<>0

2996 Parametrii şiruri/tablouri

Rutină utilizată pentru găsirea parametrilor unei variabile şir sau adresa unui număr dintr-un tablou numeric, în HL; este o subrutină a comenzii DIM pentru verificarea sintaxei.

2A52 Decupare şir

La intrare: pe stiva-calculator şi în A, B, C, D şi E parametrii şirului

2AB1 Salvare pe stiva-calculator

Rutina transferă A, B, C, D şi E pe stiva-calculator.

2ACC Expresie întreagă

Rutina returnează în BC valoarea următoarei expresii întregi şi o compară cu valoarea-limită din HL.
în A: 00 fără erori sau FF eroare

2AEE Conţinutul DE+2 în DE

Rutina plasează în DE conţinutul locaţiei a cărei adresă se află în DE.

2AF2 Obţine HL=HL*DE

Rutina verifică sintaxa şi calculează HL=HL*DE.

2AFF Comanda LET

Rutină de asignare de valori pentru LET, READ şi INPUT.

2BA6 Transferă valori

Rutina transferă valori numerice de pe stiva-calculator sau şiruri din zona de lucru în zona variabilelor BASIC.

2BAF Subrutina LET (1)

Pentru şiruri simple.

2BC6 Subrutina LET (2)

Transferul parametrilor şirurilor în zona variabilelor BASIC.

2BEA Subrutina LET (3)

Modificarea numelui unui şir.

2BF1 Transferă număr în virgulă flotantă

Transferă un număr în virgulă flotantă de pe stiva-calculator în A, B, C, D şi E.

2C02 Comanda DIM

Tablourile de numere se iniţializează cu 0.
Tablourile alfanumerice se iniţializează cu blank-uri.

2C88 Caracter alfanumeric

La ieşire: în F: C=1 dacă la intrare: în A cod caracter alfanumeric

2C8D Literal

La ieşire: în F: C=1 dacă la intrare: în A cod literă

2C9B Zecimal în virgulă flotantă

Rutina citeşte un număr zecimal cifră cu cifră şi returnează un număr zecimal în virgulă flotantă pe stiva-calculator.

2D1B Număr
La ieşire: în F: C=0 dacă la intrare: în A cod cifră
2D22 Cifră pe stiva-calculator
Cifra din A este transformată în virgulă flotantă şi transferată pe stiva-calculator.
2D28 A pe stiva-calculator
Numărul binar din A este transformat în număr în virgulă flotantă şi transferat pe stiva-calculator.
2D2B BC pe stiva-calculator
Numărul binar din BC este transformat în număr în virgulă flotantă şi transferat pe stiva-calculator.
2D3B Întreg în virgulă flotantă
Transformă un întreg dintr-o linie BASIC în număr în virgulă flotantă pe stiva-calculator.

1.9 RUTINE ARITMETICE

2D4F Format exponenţial în virgulă flotantă
Rutina transformă un număr din formatul xEy (x*10^y) în format în virgulă flotantă.
La intrare: x pe stiva-calculator
y în A
La ieşire: număr în virgulă flotantă pe stiva-calculator
2D7F Aduce întreg
Rutina încarcă în DE un număr întreg (între -65535 şi 65535) din locaţia adresată de HL (5 bytes pe stiva-calculator).
în C: semnul: 00 pentru +, FF pentru -
2D8C Memorează întreg
Rutina încarcă în memorie un număr întreg (între -65535 şi 65535) în locaţia adresată de HL (analog 2D7F).
în C: semnul: 00 pentru +, FF pentru -
2DA2 Virgulă flotantă în BC
Rutina comprimă un număr în virgulă flotantă de pe stiva-calculator în BC.
2DC1 LG (2^A)
Rutină utilizată de rutina 2DE3, calculează numărul de cifre al unui număr în virgulă flotantă
La intrare: în A exponentul numărului
La ieşire: pe stiva-calculator partea întreagă a lg(2^A)
2DD5 Virgulă flotantă în A
Rutina comprimă un număr în virgulă flotantă de pe stiva-calculator în A.
2DE3 Afişează un număr în virgulă flotantă
Rutina afişează numărul în virgulă flotantă de pe stiva-calculator.
2F8B AC=10*A+C

Subrutină a rutinei 2DE3; calculează în HL: AC=10*A+C.

2F9B Pregătire de calcul

Rutina pregăteşte un număr în virgulă flotantă pentru operaţii aritmetice (adunare, scădere, înmulţire, împărţire), efectuînd complement faţă de 2 pentru numere negative.

La ieşire: în A exponentul
primul byte: 00 pentru număr pozitiv, FF pentru număr negativ

2FBA Aduce 2 numere în virgulă flotantă

Rutina este folosită la adunarea, înmulţirea şi împărţirea numerelor în virgulă flotantă.

La ieşire: în H', B', C', C, B numărul din vîrful stivei-calculator
în L', D', E', D, E numărul al 2-lea din stiva-calculator

2FDD Shift pentru adunare

Potriveşte exponentul unui număr în virgulă flotantă pentru adunare.

3004 Adună transportul C

Rutina adună fanionul de transport C (de depăşire aritmetică), este utilizată de adunare şi înmulţire.

300F Scădere (Calculator-offset 03)

Rutina schimbă semnul scăzătorului, continuă cu adunare (3014).

3014 Adunare (Calculator-offset 0F)

30A9 HL=HL*DE

Înmulţire pe 16 biţi.

30C0 Pregătire de înmulţire în virgulă flotantă

La ieşire: în F:C=1 dacă numărul este 0
în A semnul rezultatului
înlocuieşte bitul de semn cu 1

30CA Înmulţire (Calculator-offset 04)

31AF Împărţire (Calculator-offset 05)

3214 Trunchere (Calculator-offset 3A)

Rutina rotunjeşte un număr prin eliminarea părţii fracţionare

3293 Stivuieşte 2 întregi

Rutina expandează 2 întregi in forma virgulă flotantă şi îi pune pe stivă..

3297 Stivuieşte un întreg (Calculator-offset 3D)

Rutina expandează un întreg în forma virgulă flotantă şi îl pune pe stivă..

1.10 CALCULATORUL ÎN VIRGULĂ FLOTANTĂ

Calculele în virgulă flotantă se fac prin apelarea rutinei 0028 în forma:

RST 28
DEFB *offset*
...
DEFB *end-calc*

unde DEFB semnifică operaţiile care se fac cu numerele în virgulă flotantă. Pentru operaţii se foloseşte o stivă-calculator diferită de stiva microprocesorului. Notaţia stivei-calculator: (înainte— —după), unde vîrful stivei este în dreapta (atît înainte de efectuarea operaţiei cît şi după).

32C5 Tabela de constante

		Expandat	Const.	
32C5	DEFB	00,B0,00	00,00,00,00,00	0
32CB	DEFB	40,B0,00,01	00,00,01,00,00	1
32CC	DEFB	30,00	80,00,00,00,00	1/2
32CE	DEFB	F1,49,0F,DA,A2	81,49,0F,DA,A2	PI/2
32D3	DEFB	40,B0,00,0A	00,00,0A,00,00	10

32D7 Tabela offset operaţii Calculator

	off.	adr.	nume	acţiune
32D7	00	8F, 36	jump-true	((1/0)— —) se utilizează: DEFB 00; DEFB n; începînd numărătoarea după DEFB n, unde se face saltul
32D9	01	3C, 34	exchange	(x, y— —y, x)
32DB	02	A1, 33	delete	(x— —)
32DD	03	0F, 30	substract	(x, y— —(x—y))
32DF	04	CA, 30	multyply	(x, y— —x*y)
32E1	05	AF, 31	division	(x, y— —x/y)
32E3	06	51, 38	to-power	(x, y— —x^y)
32E5	07	1B, 35	or	(x, y— —z) z=x dacă y=0; z=1 altfel (OR)
32E7	08	24, 35	no-&-no	(x, y— —z) z=x dacă y< >x; z=0 altfel (AND)
32E9	09	3B, 35	no-l-eql	(x, y— —z) z=1 dacă x< =y; z=0 altfel; comparare < = numere
32EB	0A	3B, 35	no-gr-eql	(x, y— —z) z=1 dacă x> =y; z=0 altfel; comparare > = numere
32ED	0B	3B, 35	nos-neql	(x, y— —z) z=1 dacă x< >y; z=0 altfel; comparare < > numere
32EF	0C	3B, 35	no-grtr	(x, y— —z) z=1 dacă x>y; z=0 altfel; comparare > numere
32F1	0D	3B, 35	no-less	(x, y— —z) z=1 dacă x<y; z=0 altfel; comparare < numere
32F3	0E	3B, 35	nos-eql	(x, y— —z) z=1 dacă x=y; z=0 altfel; comparare = numere
32F5	0F	14, 30	addition	(x, y— —x+y)

	off.	adr.	nume	acțiune
32F7	10	2D, 35	str-&-no	(x$, y— —z)z = x$ dacă y< >0; z = sir nul altfel; (x$ AND y)
32F9	11	3B, 35	str-l-eql	(x$, y$— —z) z = 1 dacă x$< =y$; z = 0 altfel; comparare < = șiruri
32FB	12	3B, 35	str-gr-eq	(x$,y$— —z) z = 1 dacă x$> =y$; z = 0 altfel; comparare > = șiruri
32FD	13	3B, 35	strs-neql	(x$,y$— —z) z = 1 dacă x$< >y$; z = 0 altfel; comparare < > șiruri
32FF	14	3B, 35	str-grtr	(x$,y$— —z) z = 1 dacă x$>y$; z = 0 altfel; comparare > șiruri
3301	15	3B, 35	str-less	(x$,y$— —z) z = 1 dacă x$<y$; z = 0 altfel; comparare < șiruri
3303	16	3B, 35	strs-eql	(x$,y$— —z) z = 1 dacă x$=y$; z = 0 altfel; comparare = șiruri
3305	17	9C, 35	strs-add	(x$,y$— —z$) z$ = x$ + y$; concatenare șiruri
3307	18	DE, 35	val$	(— —z) z = x$, pentru evaluare val$; VAL$ X$ se află la CH-ADD
3309	19	BC, 34	usr$	returnează în BC adresa UDG corespunzător, dacă nu eroare în A
330B	1A	45, 36	read-in	(n— —x$) citește un șir de la stream-ul *n*
330D	1B	6E, 34	negate	(x— —(-x))
330F	1C	69, 36	code	(— —z) z = codul primului caracter din șirul din CODE A$; z = 0 altfel
3311	1D	DE, 35	val	(— —z) evaluează VAL X$ ca expresie numerică; VAL X$ se află la CH-ADD
3313	1E	74, 36	len	(— —z) evaluează LEN A$, z reprezintă lungimea A$
3315	1F	B5, 37	sin	(x— —SIN x)
3317	20	AA, 37	cos	(x— —COS x)
3319	21	DA, 37	tan	(x— —TAN x)
331B	22	33, 38	asn	(x— —ASN x)
331D	23	43, 38	acs	(x— —ACS x)
331F	24	E2, 37	atn	(x— —ATN x)
3321	25	13, 37	ln	(x— —LN x)
3323	26	C4, 36	exp	(x— —EXP x)
3325	27	AF, 36	int	(x— —INT x)

	off.	adr.	nume	acţiune
3327	28	4A, 38	sqr	(x— —SQR x)
3329	29	92, 34	sgn	(x— —SGN x)
332B	2A	6A, 34	abs	(x— —ABS x)
332D	2B	AC, 34	peek	(x— —PEEK x)
332F	2C	A5, 34	in	(x— —IN x)
3331	2D	B3, 34	usr-no	(x— —) salt la rutina care începe la adresa INT x
3333	2E	1F, 36	str$	(— —z$) evaluează expresia STR$ x
3335	2F	C9, 35	chr$	(— —z$) evaluează expresia CHR$ x
3337	30	01, 35	not	(x— —z) z=1 dacă x=0; altfel z=0
3339	31	C0, 33	duplicate	(x— —x x)
333B	32	A0, 36	n-mod-m	(n, m— —(n—INT(n/m)), (INT(n/m)))
333D	33	86, 36	jump	(— —) se utilizează DEFB 33, DEFB n; unde n este byte-ul numărul n, începînd numărătoarea după DEFB n unde se face saltul relativ, n: număr cu semn în complement faţă de 2
333F	34	C6, 33	stk-data	(— —x) următorii 5 bytes după DEFB 34 reprezintă un număr x în virgulă flotantă
3341	35	7A, 36	dec-jr-nz	(— —) decrementează BREG, salt relativ dacă BREG<>0; se utilizează în forma: DEFB 35, DEFB n; unde n este byte-ul numărul n, începînd număratoarea după DEFB n unde se face saltul relativ; n număr cu semn în complement faţă de 2
3343	36	06, 35	less-0	(x— —y) y=1 dacă x<0; y=0 altfel
3345	37	F9, 34	greater-0	(x— —y) y=1 dacă x>0; y=0 altfel
3347	38	9B, 36	end-calc	(— —) reîntoarcere în rutina apelantă, orice apel RST 20 se termină cu DEFB 38
3349	39	83, 37	get-argt	(x— —v) v: valoare utilizată de funcţiile SIN şi COS; not.: y=x/(2*PI)-INT(X/(2*PI)+0.5); atunci:4*y în [-1,1]=>v=4*y; 4*y în (1,2] =>v=2-4*y; 4*y în [-2.-1) =>v=-2-4*y
334B	3A	14, 32	truncate	(x— —y) unde y este obţinut din x, prin eliminarea părţii fracţionare
334D	3B	A2, 33	fp-calc-2	rutină utilizată doar de rutina 26C9 (la adresa 2757)

	off.	adr.	nume	acţiune
334F	3C	4F, 2D	e-to-fp	(x— —y) transformă un număr din forma exponenţială (xEm, m în A) în forma virgulă flotantă y
3351	3D	97, 32	re-stack	(x— —x') x' este forma completă în virgulă flotantă a numărului x
3353	86 88 8C	49, 34	series	generează serii polinomiale Cebîşev pentru calculul funcţiilor; offset: 86 pentru SIN; 88 pentru EXP; 8C pentru LN şi ATN; după fiecare offset urmează 6, 8 respectiv 12 numere în virgulă flotantă
3355	A0 A1 A2 A3 A4	1B, 34	stk-zero stk-one stk-half stk-pi/2 stk-ten	(— —0) creează pe stivă constanta (— —1) respectivă (— —1/2) (— —PI/2) (— —10)
3357	C0 C1 C2 C3 C4 C5	2D, 34	st-mem-0 st-mem-1 st-mem-2 st-mem-3 st-mem-4 st-mem-5	(x— —) retrage un număr de pe stivă şi îl introduce în MEMBOT
3359	E0 E1 E2 E3 E4 E5	0F, 34	get-mem-0 get-mem-1 get-mem-2 get-mem-3 get-mem-4 get-mem-5	introduce un număr în stivă din MEMBOT

Observaţie: Şirurile sînt date în formă parametrică:

byte 1 nesemnificativ
bytes 2,3 adresă
bytes 4,5 lungime

335B Calculatorul

Rutină apelată de RST 28 (calculatorul în virgulă flotantă)

33A1 Delete (Calculator-offset 02)

33A2 Operaţie-unică (Calculator-offset 3B)

Rutină utilizată de calculator pentru operaţii cu un singur parametru.

33A9 Test 5 bytes

Rutina testează dacă există spaţiu suficient pe stivă (în memorie) pentru un număr între stivă şi memorie.

33B4 Număr pe stivă

Rutină utilizată de rutinele 03F8 şi 24FB.

33C0 Deplasează un număr (Calculator-offset 31)

Rutina deplasează un întreg între stivă şi memorie.

33C6 Număr pe stivă (Calculator-offset 34)

33F7 Selectează constanta
Rutină utilizată de rutina 341B
3406 Locația de memorie
Rutină de calcul a adreselor celor 6 numere în virgulă flotantă din MEMBOT.
340F Aduce din memorie (Calculator-offset E0...E5)
341B Constantă pe stivă (Calculator-offset A0...A4)
342D Stochează în memorie (Calculator-offset C0...C5)
343C Permută (Calculator-offset C0...C5)
3449 Generator de serii (Calculator-offset 86,88,8C)
Observație: instrucțiunile CALL 335E și CALL 3362 se pot citi ca RST 28.
346A Funcția ABS (Calculator-offset 2A)
346E Negare (Calculator-offset 1B)
3492 Funcția SGN (Calculator-offset 29)
34A5 Funcția IN (Calculator-offset 2C)
34AC Funcția PEEK (Calculator-offset 2B)
34B3 Funcția USR număr (Calculator-offset 2D)
34BC Funcția USR șir (Calculator-offset 19)
34E9 Test 0
La ieșire: în F: C=0 dacă numărul de pe stiva-calculator este 0
34F9 Mai mare ca 0 (Calculator-offset 37)
3501 Funcția NOT (Calculator-offset 30)
3506 Mai mic ca 0 (Calculator-offset 36)
350B 0 sau 1
Rutina setează numărul de pe stivă la valoarea:
0 dacă în F: C=0
1 dacă în F: C=1
351B Funcția OR (Calculator-offset 07)
3524 Funcția AND (Calculator-offset 08 și 10)
Rutina 3524 pentru no-&-no.
Rutina 352D pentru str-&-no.
353B Comparare (Calculator-offset 09...0E și 11...16)
359C Concatenarea șirurilor (Calculator-offset 17)
35BF Adrese stivă
La ieșire: în HL: STKEND-5=adresa primului byte al numărului de pe stiva-calculator
în DE: STKEND=adresa primului byte liber pe stiva-calculator
35C9 Funcția CHR$ (Calculator-offset 2F)
35DE Funcția VAL si VAL$ (Calculator-offset 18 și 1D)
361F Funcția STR$ (Calculator-offset 2E)
3645 Citește stream (Calculator-offset 1A)
3669 Funcția CODE șir (Calculator-offset 1C)
3674 Funcția LEN (Calculator-offset 1E)
367A Buclă cu contor (Calculator-offset 35)
3686 Salt (Calculator-offset 33)
368F Salt dacă adevărat (Calculato-offset 00)
369B Sfîrșit calcul(Calculator-offset 38)

36A0 Modul (Calculator-offset 32)
36AF Funcţia INT (Calculator-offset 27)
36C4 Funcţia EXP (Calculator-offset 26)
3713 Funcţia LN (Calculator-offset 25)
3783 Get-argt (Calculator-offset 39)
37AA Funcţia COS (Calculator-offset 20)
37B5 Funcţia SIN (Calculator-offset 1F)
37DA Funcţia TAN (Calculator-offset 21)
37E2 Funcţia ARCTAN (Calculator-offset 24)
3833 Funcţia ARCSIN (Calculator-offset 22)
3843 Funcţia ARCCOS (Calculator-offset 23)
384A Funcţia SQR (Calculator-offset 28)
3851 Funcţia putere (Calculator-offset 06)
386E Neutilizat
3D00 Matricea caracterelor
3D00-3DFF Fiecărui caracter afişabil de la 20 (blank) la 7F (copyright) i se atribuie 8 bytes care definesc matricea 8*8 pixeli cu ajutorul căreia se afişează..

1.11 VARIABILELE DE SISTEM

Notaţie:
- m variabila poate fi modificată
- N variabila nu trebuie modificată
- - variabilă modificată permanent de sistem
- [ ] în paranteze sînt adresele rutinelor care le utilizează

Adresă	Byte	Mod	Nume	Observaţii
5C00	8	N	KSTATE	defineşte starea tastaturii [02BF, 11B7]
5C08	1	-	LAST-K	conţine codul ultimei taste apăsate (02BF)
5C09	1	m	REPDEL	durata maximă de apăsare a unei taste fără repetare este 0.02s*REPDEL [02BF, 11B7]
5C0A	1	m	REPPER	durata între repetiţii cînd o tastă este menţinută apăsată (0.1s) [0310]
5C0B	2	-	DEFADD	adresa argumentelor FN [27BD]
5C0D	1	-	K-DATA	byte de control, culori tastate [10A8]
5C0E	2	-	TVDATA	byte de control, culoare utilizată de AT şi TAB [0A60]
5C10	38	N	STRMS	conţine adresele canalelor
5C36	2	m	CHARS	conţine adresa tabelului matrice al caracterelor (3D00-0100=3C00)
5C38	1	m	RASP	durata semnalului sonor
5C39	1	m	PIP	lungimea semnalului sonor la apăsarea unei taste

5C3A	1	m	ERR-NR	codul erorii întîlnite este ERR-NR + 1, în absența erorii ERR-NR = FF [0053, 2089]
5C3B	1	N	FLAGS	fanioane [02BF, 0A23, 0A4F, 0A6D, 0ADC, 0C55, 11B7, 1795, 1F3A, 1F60, 2089, 2AEE, 35DE]
5C3C	1	N	TV-FLAGS	fanioane [0970, 0ADC, 0C55, 0D4D, 0E44, 11B7, 1795, 2089]
5C3D	1	N	ERR-SP	adresa stivei în caz de eroare [0F2C, 11B7, 1EAC, 1EED, 1F23, 2089]
5C3F	2	-	LIST-SP	conține adresa de retur după LIST [0C55, 1795]
5C41	1	-	MODE	modul cursor: K,L,C,E sau G [02BF, 0F2C, 0F81, 18E1]
5C42	2	m	NEWPPC	numărul liniei BASIC unde trebuie efectuat un salt [0808, 1D03, 1E67]
5C44	2	m	NSPPC	numărul instrucțiunii dintr-o linie BASIC unde trebuie efectuat un salt [0808, 1D03, 1E67]
5C45	2	m	PPC	numărul liniei BASIC în curs de execuție [1D03, 1EED]
5C47	1	m	SUBPPC	numărul instrucțiunii în curs de execuție din linia BASIC [1D03, 1EED]
5C48	1	m	BORDCR	culoarea border-ului și a liniilor de jos ale ecranului [053F, 0D4D, 0E44]
5C49	2	m	E-PPC	numărul liniei în care se găsește cursorul [0FA9, 1795]
5C4B	2	N	VARS	adresa zonei de memorie în care se găsesc variabilele BASIC [0605, 0808, 08B6, 11B7, 1EAC]
5C4D	2	-	DEST	adresa variabilei în curs de asignare [2AFF]
5C4F	2	N	CHANS	adresa datelor asupra diferitelor canale
5C51	2	N	CURCHL	adresa de informații in/out [0AD6, 0D6B, 0DAF, 0FA9, 361F]
5C53	2	N	PROG	adresa de început a programului BASIC [0605, 08B6, 092C, 11B7, 196E]
5C55	2	N	NXTLIN	adresa următoarei linii BASIC de executat [1BB2, 1D03]
5C57	2	N	DATADD	adresa sfîrșitului ultimei date (DATA) [11B7, 1DEC, 1E42]
5C59	2	N	E-LINE	conține adresa zonei editor [0605, 0808, 11B7, 19FB, 2AEE, 2C02]
5C5B	2	m	K-CUR	adresa cursorului pe ecran [0F2C, 0F81, 18E1, 20B9, 361F]
5C5D	2	N	CH-ADD	adresa următorului caracter de interpretat [0018, 0074, 19FB, 1D03, 2089, 35DE]
5C5F	2	m	X-PTR	adresa caracterului care precede cursorul de eroare în editare sau INPUT [0808, 092C, 1DEC, 2089]
5C61	2	N	WORKSP	adresa zonei de lucru [0030, 11B7, 169E 2089]
5C63	2	N	STKBOT	adresa de început a stivei-calculator [11B7, 169E, 2089]

5C65	2	N	STKEND	adresa de sfîrşit a stivei-calculator [11B7, 19FB, 1EAC, 1F05]
5C67	1	-	BREG	contor utilizat de o instrucţiune pseudo-DJNZ [0028]
5C68	2	-	MEM	adresa stivei-calculator [1D03]
5C6A	1	m	FLAGS2	fanioane pentru decodarea tastaturii [0333, 0DAF, 1795, 18E1, 2089]
5C6B	1	N	DF-SZ	numărul liniilor din partea de jos a ecranului [0C55, 0D6B, 11B7, 1795, 2089]
5C6C	2	m	S-TOP	numărul liniei superioare într-un LIST, LLIST
5C6E	2	m	OLDPPC	numărul liniei BASIC de unde se continuă programul după CONTINUE
5C70	1	m	OSPPC	numărul instrucţiunii într-o linie de unde se continuă programul după CONTINUE
5C71	1	-	FLAGX	fanioane [0F2C, 0FA9, 2089, 2AEE]
5C72	2	-	STRLEN	numărul caracterelor de interpretat [1D03, 2AEE]
5C74	2	-	T-ADDR	adresa următoarei rubrici în tabela de sintaxă, utilizată de rutinele SAVE şi LOAD
5C76	2	m	SEED	baza seriei numerelor pseudo-aleatoare [1E4F]
5C78	3	m	FRAMES	control de ceas real, FRAMES*0.02s= timpul scurs de la pornirea calculatorului (sau reset hard) [0038, 1E4F]
5C7B	2	m	UDG	adresa primului caracter grafic
5C7D	1	m	COORDS	coordonata x a ultimului pixel afişat [22DC, 2320, 2382]
5C7E	1	m	COORDS	coordonata y a ultimului pixel afişat [22DC, 2320, 2382]
5C7F	1	m	P-POSN	poziţia capului de imprimantă
5C80	1	m	PR-CC	adresa următorului caracter de imprimat
5C81	1	m	-	neutilizat
5C82	2	m	ECHO-E	numărul ultimei coloane utilizată la un INPUT [0ADC]
5C84	2	m	DF-CC	adresa următoarei poziţii de scriere PRINT [0ADC]
5C86	2	m	DFCCL	adresa următoarei poziţii de scriere în partea de jos a ecranului
5C88	1	N	S-POSN	numărul coloanei următoarei instrucţiuni PRINT
5C89	1	N	S-POSN	numărului liniei următoarei instrucţiuni PRINT
5C8A	2	N	SPOSNL	numărul coloanei şi liniei de afişare pentru partea de jos a ecranului
5C8C	1	m	SCR-CT	contor al numărului de linii inainte de scroll? [0C55, 0DAF, 2089]
5C8D	1	m	ATTR-P	numărul culorii permanente [0DAD, 0D6B, 0E44, 11B7]
5C8E	1	m	MASK-P	culori transparente permanente
5C8F	1	-	ATTR-T	numărul culorii temporare [0BDB, 0C55, 0D4D, 11B7, 18C1]

5C90	1	-	MASK-T	culori transparente temporare
5C91	1	m	P-FLAG	fanioane [0A3D, 0BD3, 0C55, 18C1, 22DC]
5C92	30	N	MEMBOT	6 locații-registre pentru numere în virgulă flotantă (5 bytes fiecare) [0028]
5CB0	2	m	NMIADD	inutilizată [0066]
5CB2	2	m	RAMTOP	adresa ultimului byte utilizat de BASIC [11B7, 1EAC]
5CB4	2	m	P-RAMT	adresa ultimului byte fizic

1.12 HARTA MEMORIEI TOTALE

0000	ROM
4000	Harta de pixeli
5800	Atribute
5B00	Tampon de imprimantă
5C00	Variabile de sistem
5CB6	Harta microdrive
CHANS	Informații de canale
PROG	Program BASIC
VARS	Variabile
E-LINE	Comanda sau linia BASIC în curs de editare
WORKSP	Input date
	Spațiu de lucru temporar
STKBOT	Stiva-calculator
STKEND	Liber
SP-register	Stiva microprocesorului
	Liber
	Stiva GO SUB
RAMTOP	UDG
FFFF	-

1.13 STRUCTURI BASIC

Structura liniei BASIC

byte-inf	byte-sup	lungime-text+1	text	cod Enter
număr de	linie			

Observație: Numărul de linie pe 2 bytes este singurul pentru care nu este

respectată convenţia Intel (byte-ul mai puţin semnificativ la adresa inferioară şi byte-ul mai semnificativ la adresa superioară).

Variabilă numerică, nume = 1 literă

011	litera-1—60	exponent	bit-semn, mantisă = 4 bytes

Variabilă numerică, nume = k litere (k > 1)

101	litera-1—60	litera-2	...	1	litera-k	valoare = 5 bytes

Variabilă tablou numerică, k dimensiuni

1 0 0	litera-1-60	lungime-totală-elemente + dimensiuni + 1 = 2bytes

numărul dimensiunilor	dimensiune-1 = 2 bytes	...

dimensiune-k = 2 bytes	valori = 5 bytes fiecare

Ordinea este x(1,1...), x(2,1...) ... (invers lexicografică).

Variabilă şir, 1 dimensiune

010	litera-60	lungime = 2 bytes	text

Variabilă tablou şir, k dimensiuni

1 0 0	litera-1-60	lungime-totală-elemente + dimensiuni + 1 = 2 bytes

numărul dimensiunilor	dimensiune-1 = 2 bytes	...

dimensiune-k = 2 bytes	elemente = 1 byte fiecare

Variabilă de control pentru FOR-NEXT

111	litera—60	valoare = 5 bytes	limita = 5 bytes	treapta = 5 bytes

byte-inf	byte-sup	număr-instrucţiune
număr linie	FOR	în linie FOR

1.14 SETUL DE CARACTERE SPECTRUM

Observaţie: corespunde cu codurile ASCII între 32 şi 127

	0	1	2	3	4	5	6	7	8	9
0							PRINT comma	EDIT control	cursor stînga	cursor dreapta
10	cursor jos	cursor sus	Delete	Enter	number		INK control	PAPER control	FLASH control	BRIGHT control
60		=		?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	–	£	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	copy-right	udg-1	udg-2
130	udg-3	udg-4	udg-5	udg-6	udg-7	udg-8	udg-9	udg-10	udg-11	udg-12
140	udg-13	udg-14	udg-15	udg-16	udg-A	udg-B	udg-C	udg-D	udg-E	udg-F
150	udg-G	udg-H	udg-I	udg-J	udg-K	udg-L	udg-M	udg-N	udg-O	udg-P
160	udg-Q	udg-R	udg-S	udg-T	udg-U	RND	INKEY$	PI	FN	POINT
170	SCRN.	ATTR	AT	TAB	VAL$	CODE	VAL	LEN	SIN	COS
180	TAN	ASN	ACS	ATN	LN	EXP	INT	SQR	SGN	ABS
190	PEEK	IN	USR	STR$	CHR$	NOT	BIN	OR	AND	< =
200	> =	< >	LINE	THEN	TO	STEP	DEF FN	CAT	FOR-. MAT	MOVE
210	ERASE	OPEN #	CLOSE #	MERGE	VERIFY	BEEP	CIRCLE	INK	PAPER	FLASH
220	BRIGHT	INVER-SE	OVER	OUT	LPRINT	LLIST	STOP	READ	DATA	RESTO-RE
230	NEW	BOR-DER	CONTI-NUE	DIM	REM	FOR	GO TO	GO SUB	INPUT	LOAD
240	LIST	LET	PAUSE	NEXT	POKE	PRINT	PLOT	RUN	SAVE	RAND.
250	IF	CLS	DRAW	CLEAR	RE-TURN	COPY				

2 ASAMBLORUL/DEZASAMBLORUL

2.1 MICROPROCESORUL Z80

Regiştrii microprocesorului Z80 sînt:

A acumulator (registru principal) (8 biţi)

F flags (registru fanioane acumulator) (8 biţi) conţine:

bit 7	S semn	bit 3	* nedefinit
bit 6	Z zero	bit 2	P/V paritate/depăşre
bit 5	* nedefinit	bit 1	N adunare/scădere
bit 4	H half carry	bit 0	C carry

B,C,D,E,H,L regiştri de uz general (8 biţi)
pot fi utilizaţi în pereche BC, DE, HL pentru a forma o adresă

A, F, B, C, D, E, H, L set alternat de regiştri; operează la fel ca analogii lor

IX, IY regiştrii index(16 biţi)

SP stack-pointer: adresa vîrfului stivei care creşte în jos (16 biţi)

PC program-counter: adresa următoarei instrucţiuni (16 biţi)

R, I regiştri cu destinaţie specială (8 biţi)

Notaţii:

r,s	regiştri (8 biţi)
rp	pereche de regiştri (16 biţi)
d,n,e	dată numerică (0-FF) (8 biţi)
nn	dată numerică (0-FFFF) (16 biţi)
b	bit (7-0)
Fc	bitul C din F
Fz	bitul Z din F
c	condiţie
T	perioadă de tact, ceas

fanion în F: ! afectat conform rezultatului
. neafectat
* nedefinit

<-> permutarea conţinutului a 2 regiştri

<- asignare
(rp) conţinutul byte-ului de memorie a cărui adresă se află în rp
(nn) conţinutul byte-ului de memorie a cărui adresă este nn
A0~A15 magistrala de adrese
D0~D7 magistrala de date
Convenţia Intel: byte-ul mai puţin semnificativ la adresa inferioară
byte-ul mai semnificativ la adresa superioară

2.2 ÎNCĂRCARE PE 8 BIŢI

Format	Operaţie	Flags	Cod	T	Observaţii
LD r,s	r <- s	..*.*...	01-r- -s-	4	r,s Reg
LD r,n	r <- n	..*.*...	00-r-100 -n-	7	000 B 001 C
LD r, (HL)	r <- (HL)	..*.*...	01-r-110	7	010 D
LD r, (IX+d)	r <- (IX+d)	..*.*...	11011101 01-r-110 -d-	19	011 E 100 H 101 L
LD r, (IY+d)	r <- (IY+d)	..*.*...	11111101 01-r-110 -d-	19	111 A
LD (HL), n	(HL) <- n	..*.*...	00110110 -n-	10	
LD (IX+d), n	(IX+d) <- n	..*.*...	11011101 00110110 -d- -n-	19	
LD (IY+d), n	(IY+d) <- n	..*.*...	11111101 00110110 -d- -n-	19	
LD A, (BC)	A <- (BC)	..*.*...	00001010	7	
LD A, (DE)	A <- (DE)	..*.*...	00011010	7	
LD A, (nn)	A <- (nn)	..*.*...	00111010 -n- -n-	13	
LD (BC), A	(BC) <- A	..*.*...	00000010	7	
LD (DE), A	(DE) <- A	..*.*...	00010010	7	
LD (nn), A	(nn) <- A	..*.*...	00110010 -n- -n-	13	

Format	Operație	Flags	Cod	T	Observații
LD A, I	A <- I	!!*0b0.	11101101 01010111	9	b = IFF, bit de întrerupere
LD A, R	A <- R	!!*0*b0.	11101101 01011111	9	
LD I, A	I <- A	..*.*...	11101101 01000111	9	
LD R, A	R <- A	..*.*...	11101101 01001111	9	
LD (HL), r	(HL) <- r	..*.*...	01110-r-	7	
LD (IX+d), r	(IX+d) <- r	..*.*...	11011101 01110-r- -d-	19	
LD (IY+d), r	(IY+d) <- r	..*.*...	11111101 01110-r- -d-	19	

2.3 ÎNCĂRCARE PE 16 BIȚI

Format	Operație	Flags	Cod	T	Observații
LD rp, nn	rp <- nn	..*.*...	00rp001 -n- -n-	10	rp Reg.p. 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX <- nn	..*.*...	11011101 00100001 -n- -n-	14	
LD IY, nn	IY <- nn	..*.*...	11111101 00100001 -n- -n-	14	
LD HL, (nn)	H <- (nn+1) L <- (nn)	..*.*...	00101010 -n- -n-	16	
LD rp, (nn)	rp_h <- (nn+1) rp_l <- (nn)	..*.*...	11101101 01rp1011 -n- -n-	20	

Format	Operaţie	Flags	Cod	T	Observaţii
LD IX,(nn)	IX_h <- (nn+1) IX_l <- (nn)	..*.*...	11011101 00101010 -n- -n-	20	
LD IY, (nn)	IY_h <- (nn+1) IY_l <- (nn)	..*.*...	11111101 00101010 -n- -n-	20	
LD (nn), HL	(nn+1) <- H (nn) <- L	..*.*...	00100010 -n- -n-	16	
LD (nn), rp	(nn+1) <- rp_h (nn) <- rp_l	..*.*...	11101101 01rp0011 -n- -n-	20	
LD (nn), IX	(nn+1) <- IX_h (nn) <- IX_l	..*.*...	11011101 00100010 -n- -n-	20	
LD (nn),IY	(nn+1) <- IY_h (nn) <- IY_l	..*.*...	11111101 00100010 -n- -n-	20	
LD SP, HL	SP <- HL	..*.*...	11111001	6	
LD SP,IX	SP <- IX	..*.*...	11011101 11111001	10	
LD SP, IY	SP <- IY	..*.*...	11111101 1111101	10	
PUSH rp	(SP-2) <- rp_l (SP-1) <- rp_h SP <- SP-2	..*.*...	11rp0101	11	
PUSH IX	(SP-2) <- IX_h (SP-1) <- IX_l SP <- SP-2	..*.*...	11011101 11100101	15	
PUSH IY	(SP-2) <- IY_h (SP-1) <- IY_l SP <- SP-2	..*.*...	11111101 11100101	15	
POP rp	rp_h <- (SP+1) rp_l <- (SP) SP <- SP+2	..*.*...	11rp0001	10	
POP IX	IX_h <- (SP+1) IX_l <- (SP) SP <- SP+2	..*.*...	11011101 11100001	14	
POP IY	IY_h <- (SP+1) IY_l <- (SP) SP <- SP+2	..*.*...	11111101 11100001	14	

2.4 SCHIMB, TRANSFER DE BLOCURI ŞI CĂUTARE

Format	Operaţie	Flags	Cod	T	Observaţii
EX DE, HL	DE <-> HL	..*.*...	11101011	4	comutare pe setul
EX AF,AF'	AF <-> AF'	!!*!*!!!	00001000	4	alternat de
EXX	BC <-> BC' DE <-> DE' HL <-> HL'	..*.*...	11011001	4	regiştri
EX (SP), HL	H <-> (SP + 1) L <-> (SP)	..*.*...	11100011	19	
EX (SP),IX	IXh <-> (SP + 1) IXl <-> (SP)	..*.*...	11011101 11100011	23	
EX (SP),IY	IYh <-> (SP + 1) IYl <-> (SP)	..*.*...	11111101 11100011	23	
LDI	(DE) <- (HL) DE <- DE + 1 HL <- HL + 1 BC <- BC-1	..*0*!0.	11101101 10100000	16	P/V = 0 < = > BC-1 = 0; altfel P/V = 1
LDIR	LDI pina cind BC = 0	..*0*00.	11101101 10110000	21 16	dacă BC < > 0 dacă BC = 0
LDD	(DE) <- (HL) DE <- DE-1 HL <- HL-1 BC <- BC-1	..*0*!0.	11101101 10101000	16	P/V = 0 < = > BC-1 = 0; altfel P/V = 1
LDDR	LDI pînă cînd BC = 0	..*0*00.	11101101 10111000	21 16	dacă BC < > 0 dacă BC = 0
CPI	A comparat cu (HL) HL <- HL + 1 BC <- BC-1	!!*!*!1.	11101101 10100001	16	Z = 1 dacă A = (HL); P/V = 0 dacă BC-1 = 0
CPIR	CPI pînă cînd A = (HL) sau BC = 0	!!*!*!1.	11101101 10110001	21 . 16	dacă A < > (HL) şi BC < >0 dacă A = (HL) sau BC = 0 Z, P/V conform CPI
CPD	A comparat cu (HL) HL <- HL-1 BC <- BC-1	!!*!*!1.	11101101 10101001	16	Z = 1 dacă A = (HL); P/V = 0 dacă BC-1 = 0

Format	Operație	Flags	Cod	T	Observații
CPDR	CPD pînă cînd A=(HL) sau BC=0	!!*!*!1.	11101101 10111001	21 - 16	dacă A< >(HL) și BC< >0 dacă A=(HL) sau BC=0 Z, P/V conform CPI

2.5 ARITMETICĂ ȘI LOGICĂ PE 8 BIȚI

Format	Operație	Flags	Cod	T	Observații
ADD A, r	A <- A+r	!!*!*V0!	10000-r-	4	r Reg.
ADD A, n	A <- A+n	!!*!*V0!	11000110 -n-	7	000 B 001 C
ADD A, (HL)	A <- A+(HL)	!!*!*V0!	10000110	7	010 D
ADD A, (IX+d)	A <- A+(IX+d)	!!*!*V0!	11011101 10000110 -d-	19	011 E 100 H 101 L
ADD A, (IY+d)	A <- A+(IY+d)	!!*!*V0!	11111101 10000110 -d-	19	111 A
ACD A, r	A <- A+r+Fc	!!*!*V0!	10001-r-	4	
ACD A, n	A <- A+n+Fc	!!*!*V0!	11001110 -n-	7	
ACD A, (HL)	A <- A+(HL)+Fc	!!*!*V0!	10001110	7	
ACD A, (IX+d)	A <- A+(IX+d)+Fc	!!*!*V0!	11011101 10001110 -d-	19	
ACD A, (IY+d)	A <- A+(IY+d)+Fc	!!*!*V0!	11111101 10001110 -d-	19	
SUB r	A <- A—r	!!*!*V1!	10010-r-	4	
SUB n	A <- A—n	!!*!*V1!	11010110 -n-	7	
SUB (HL)	A <- A—(HL)	!!*!*V1!	10010110	7	
SUB (IX+d)	A <- A—(IX+d)	!!*!*V1!	11011101 10010110 -d-	19	
SUB (IY+d)	A <- A—(IY+d)	!!*!*V1!	11111101 10010110 -d-	19	

Format	Operație	Flags	Cod	T	Observații
SBC A, r	A <- A—r—F_c	!!*!*V1!	10011-r- 11011110 -n-	4	
SBC A, n	A <- A—n—F_c	!!*!*V1!	11011110 -n-	7	
SBC A, (HL)	A <- A—(HL)—F_c	!!*!*V1!	10011110	7	
SBC A, (IX+d)	A <- A—(IX+d)—F_c	!!*!*V1!	11011101 10011110 -d-	19	
SBC A, (IY+d)	A <- A—(IY+d)—F_c	!!*!*V1!	11111101 10011110 -d-	19	
AND r	A <- A și r	!!*1*P00	10100-r-	4	
AND n	A <- A și n	!!*1*P00	11100110 -n-	7	
AND (HL)	A <- A și (HL)	!!*1*P00	10100110	7	
AND (IX+d)	A <- A și (IX+d)	!!*1*P00	11011101 10100110 -d-	19	
AND (IY+d)	A <- A și (IY+d)	!!*1*P00	11111101 10100110 -d-	19	
OR r	A <- A sau r	!!*1*P00	10110-r-	4	
OR n	A <- A sau n	!!*1*P00	11110110 -n-	7	
OR (HL)	A <- A sau (HL)	!!*1*P00	10110110	7	
OR (IX+d)	A <- A sau (IX+d)	!!*1*P00	11011101 10110110 -d-	19	
OR (IY+d)	A <- A sau (IY+d)	!!*1*P00	11111101 10110110 -d-	19	
XOR r	A <- A sau-exclusiv r	!!*1*P00	10101-r-	4	
XOR n	A <- A sau-exclusiv n	!!*1*P00	11101110 -n-	7	
XOR (HL)	A <- A sau-exclusiv (HL)	!!*1*P00	10101110	7	
XOR (IX+d)	A <- A sau-exclusiv (IX+d)	!!*1*P00	11011101 10101110 -d-	19	
XOR (IY+d)	A <- A sau-exclusiv (IY+d)	!!*1*P00	11111101 10101110 -d-	19	
CP r	A comparat cu r	!!*!*V1!	10111-r-	4	
CP n	A comparat cu n	!!*!*V1!	11111110 -n-	7	
CP (HL)	A comparat cu (HL)	!!*!*V1!	10111110	7	

Format	Operaţie	Flags	Cod	T	Observaţii
CP (IX+d)	A comparat cu (IX+d)	!!*!*V1!	11011101 1011110 -d-	19	
CP (IY+d)	A comparat cu (IY+d)	!!*!*V1!	11111101 1011110 -d-	19	
INC r	r <- r+1	!!*!*V0.	00-r-100	4	
INC (HL)	(HL) <- (HL)+1	!!*!*V0.	00110100	11	
INC (IX+d)	(IX+d) <- (IX+d)+1	!!*!*V0.	11011101 00110100 -d-	23	
INC (IY+d)	(IY+d) <- (IY+d)+1	!!*!*V0.	11111101 00110100 -d-	23	
DEC r	r <- r—1	!!*!*V0.	00-r-101	4	
DEC (HL)	(HL) <- (HL)—1	!!*!*V0.	00110101	11	
DEC (IX+d)	(IX+d) <- (IX+d)—1	!!*!*V0.	11011101 00110101 -d-	23	
DEC (IY+d)	(IY+d) <- (IY+d)—1	!!*!*V0.	11111101 00110101 -d-	23	

2.6 CONTROL

Format	Operaţie	Flags	Cod	T	Observaţii
DAA	ajustare zecimală	!!*!*P.!	00100111	4	bit 3-0: dacă 9 sau H=1 adaugă 6 bit 4-7: dacă 9 sau C=1 adaugă 6+H
CPL	A <- neg A	..*1*.1.	00101111	4	complement faţă de 1
NEG	A <- (neg A)+1	!!*!*V1!	11101101	8	complement faţă de 2
CCF	Fc <- neg Fc	..***.0!	00111111	4	
SCF	Fc <- 1	..*.*...	00110111	4	
NOP	nici o operaţie	..*.*...	00000000	4	

Format	Operaţie	Flags	Cod	T	Observaţii
HALT	microprocesor stop	..*.*...	01110110	4	aşteaptă întrerupere sau restart hard
DI	IFF <- 0	..*.*...	11110011	4	dezactivează întreruperile mascabile
EI	IFF <- 1	..*.*...	11111011	4	activează întreruperile mascabile
IM0	mod 0 întreruperi	..*.*...	11101101 01000110	8	
IM1	mod 1 întreruperi	..*.*...	11101101 01010110	8	
IM2	mod 2 întreruperi	..*.*...	11101101 01011110	8	

2.7 ARITMETICĂ PE 16 BIŢI

Format	Operaţie	Flags	Cod	T	Observaţii
ADC HL, ss	HL <- HL+ss	..***.0!	00ss1001	11	ss Reg.
ADC HL, ss	HL <- HL+ss+Fc	!!***V0!	11101101 01ss1010	15	00 BC 01 DE
SBC HL, ss	HL <- HL—ss—Fc	!!***V1!	11101101	15	10 HL 11 SP
ADD IX, pp	IX <- IX+pp	..***.0!	11011101 00pp1001	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY <- IY+rr	..***.0!	11111101 00rr1001	15	rr Reg. 00 BC 01 DE 10 IY 11 SP
INC ss	ss <- ss+1	..*.*...	11ss1011	6	
INC IX	IX <- IX+1	..*.*...	11011101 00100011	10	
INC IY	IY <- IY+1	..*.*...	11111101 00100011	10	
DEC ss	ss <- ss—1	..*.*...	00ss1011	6	
DEC IX	IX <- IX—1	..*.*...	11011101 00101011	10	
DEC IY	IY <- IY—1	..*.*...	11111101 00101011	10	

2.8 ROTAȚIE ȘI SHIFT

Format	Operație	Flags	Cod	T	Observații
RLCA	shift A spre stînga bit 0(A); Fc <- bit 7(A)	..*0*.0!	00000111	4	
RLA	shift A spre stînga Fc <- bit 7(A) bit 0(A) <- Fc	..*0*.0!	00010111	4	
RRCA	shift A spre dreapta bit 7(A); Fc <- bit 0(A)	..*0*.0!	00001111	4	
RRA	shift A spre dreapta bit 7(A) <- Fc	..*0*.0!	00011111	4	
RLC r	shift r spre stînga bit 0(r); Fc <- bit 7(r)	!!*0*P0!	11001011 00000-r-	8	r Reg. 000 B
RLC (HL)	shift (HL) spre stînga bit 0((HL)); Fc <- bit 7((HL))	!!*0*P0!	11001011 00000110	15	001 C 010 D 011 E
RLC (IX+d)	shift (IX+d) spre stînga bit 0((IX+d)); Fc <- bit 7((IX+d))	!!*0*P0!	11011101 11001011 -d-	23	100 H 101 L 111 A
RLC (IY+d)	shift (IY+d) spre stînga bit 0((IY+d)); Fc <- bit 7((IY+d))	!!*0*P0!	11111101 11001011 -d-	23	
RL r	shift r spre stînga Fc <- bit 7(r) bit 0(r) <- Fc	!!*0*P0!	11001011 00010-r-	8	
RL (HL)	shift (HL) spre stînga Fc <- bit 7((HL)) bit 0((HL)) <- Fc	!!*0*P0!	11001011 00010110	15	
RL (IX+d)	shift (IX+d) spre stînga Fc <- bit 7((IX+d)) bit 0((IX+d)) <- Fc	!!*0*P0!	11011101 11001011 -d-	23	
RL (IY+d)	shift (IY+d) spre stînga Fc <- bit 7((IY+d)) bit 0((IY+d)) <- Fc	!!*0*P0!	11111101 11001011 -d-	23	
RRC r	shift r spre dreapta bit 7(r); Fc <- bit 0(r)	!!*0*P0!	11001011 00001-r-	8	
RRC (HL)	shift (HL) spre dreapta bit 7((HL)); Fc <- bit 0((HL))	!!*0*P0!	11001011 00001110	15	

Format	Operație	Flags	Cod	T	Observații
RRC (IX+d)	shift (IX+d) spre dreapta bit 7((IX+d)); F_c <- bit 0((IX+d))	!!*0*P0!	11011101 11001011 -d- 00001110	23	
RRC (IY+d)	shift (IY+d) spre dreapta bit 7((IY+d)); F_c <- bit 0((IY+d))	!!*0*P0!	11111101 11001011 -d- 00001110	23	
RR r	shift r spre dreapta bit 7(r) <- F_c	!!*0*P0!	11001011 00011-r-	8	
RR (HL)	shift (HL) spre dreapta bit 7((HL)) <- F_c	!!*0*P0!	11001011 00011110	15	
RR (IX+d)	shift (IX+d) spre dreapta bit 7((IX+d)) <- F_c	!!*0*P0!	11011101 11001011 -d- 00011110	23	
RR (IY+d)	shift (IY+d) spre dreapta bit 7((IY+d)) <- F_c	!!*0*P0!	11111101 00011110 -d- 00011110	23	
SLA r	shift r spre stînga F_c <- bit 7(r) bit 0(r) <- 0	!!*0*P0!	11001011 00100-r-	8	
SLA (HL)	shift (HL) spre stînga F_c <- bit 7((HL)) bit 0((HL)) <- 0	!!*0*P0!	11001011 00100110	15	
SLA (IX+d)	shift (IX+d) spre stînga F_c <- bit 7((IX+d)) bit 0((IX+d)) <- 0	!!*0*P0!	11011101 11001011 -d- 00100110	23	
SLA (IY+d)	shift (IY+d) spre stînga F_c <- bit 7((IY+d)) bit 0(IY+d)) <- 0	!!*0*P0!	11111101 11001011 -d- 00100110	23	
SRA r	shift r spre stînga F_c <- bit 0(r) bit 7(r) nu se modifică	!!*0*P0!	11001011 00101-r-	8	
SRA (HL)	shift (HL) spre stînga F_c <- bit 0((HL)) bit 7((HL)) nu se modifică	!!*0*P0!	11001011 00101110	15	
SRA (IX+d)	shift (IX+d) spre stînga F_c <- bit 0((IX+d)) bit 7((IX+d)) nu se modifică	!!*0*P0!	11011101 11001011 -d- 00101110	23	

Format	Operație	Flags	Cod	T	Observații
SRA (IY+d)	shift (IY+d) spre stînga F_c <- bit 0((IY+d)) bit 7((IY+d)) nu se modifică	!!*0*P0!	11111101 11001011 -d- 00101110	23	
SRL r	shift r spre dreapta F_c <- bit 0(r) bit 7(r) <- 0	!!*0*P0!	11001011 00111-r-	8	
SRL (HL)	shift (HL) spre dreapta F_c <- bit 0((HL)) bit 7((HL)) <- 0	!!*0*P0!	11001011 00111110	15	
SRL (IX+d)	shift (IX+d) spre dreapta F_c <- bit 0((IX+d)) bit 7((IX+d)) <- 0	!!*0*P0!	11011101 11001011 -d- 00111110	23	
SRL (IY+d)	shift (IY+d) spre dreapta F_c <- bit 0((IY+d)) bit 7((IY+d)) <- 0	!!*0*P0!	11111101 11001011 -d- 00111110	23	
RLD	bit0-3(HL) <-bit0-3(A) bit4-7(HL) <-bit0-3(HL) bit0-3(A) <-bit4-7(HL) permutare circulară	!!*0*P0.	11101101 01101111	18	
RRD	bit0-3(HL) <-bit4-7(HL) bit4-7(HL) <-bit0-3(A) bit0-3(A) <-bit0-3(HL)	!!*0*P0.	11101101 01100111	18	

2.9 SET, RESET ȘI TEST PE BIȚI

Format	Operație	Flags	Cod	T	Observații
BIT b, r	F_z <- r_b	*!*1*.0.	11001011 01-b- -r-	8	r Reg. 000 B
BIT b, (HL)	F_z <- $(HL)_b$	*!*1*.0.	11001011 01-b-110	12	001 C 010 D
BIT b, (IX+d)	F_z <- $(IX+d)_b$	*!*1*.0.	11011101 11001011 -d- 01-b-110	10	011 E 100 H 101 L 111 A

Format	Operaţie	Flags	Cod	T	Observaţii
BIT b, (IY+d)	F_z <- $(IY+d)_b$	*!*1*.0.	11111101 11001011 -d- 01-b-110	20	b Bit. 000 0 001 1 010 2 011 3 100 4 101 5 110 6 111 7
SET b, r	r_b <- 1	..*.*...	11001011 11-b- -r-	8	
SET b, (HL)	$(HL)_b$ <- 1	..*.*...	11001011 11-b-110	15	
SET b, (IX+d)	$(IX+d)_b$ <- 1	..*.*...	11011101 11001011 -d- 11-b-110	23	
SET b, (IY+d)	$(IY+d)_b$ <- 1	..*.*...	11111101 11001011 -d- 11-b-110	23	
RES b, r	r_b <- 0	..*.*...	11001011 10-b- -r-	8	
RES b, (HL)	$(HL)_b$ <- 0	..*.*...	11001011 10-b-110	15	
RES b, (IX+d)	$(IX+d)_b$ <- 0	..*.*...	11011101 11001011 -d- 10-b-110	23	
RES b, (IY+d)	$(IY+d)_b$ <- 0	..*.*...	11111101 11001011 -d- 10-b-110	23	

2.10 TRANSFER

Format	Operaţie	Flags	Cod	T	Observaţii
JP nn	PC <- nn	..*.*...	11000011 -n- -n-	10	c Cond.în F 000 NZ (not 0) 001 Z (=0) 010 NC (C=0) 011 C (C=1) 100 PO (impar) 101 PE (par) 110 P (pozitiv) 111 M (negativ)
JP c, nn	dacă c adevărat atunci PC <- nn	..*.*...	11-c-010 -n- -n-	10	

Format	Operaţie	Flags	Cod	T	Observaţii
JR e	PC <- PC+e	..*.*...	00011000 -e-	12	e număr cu semn în complement faţă de 2
JR C, e	dacă C=0 continuă altfel PC <- PC+e	..*.*...	00111000 e-2	7 12	dacă C=0 dacă C=1
JR NC, e	dacă C=1 continuă altfel PC <- PC+e	..*.*...	00110000 e-2	7 12	dacă C=1 dacă C=0
JR Z, e	dacă Z=0 continuă altfel PC <- PC+e	..*.*...	00101000 e-2	7 12	dacă Z=0 dacă Z=1
JR NZ, e	dacă Z=1 continuă altfel PC <- PC+e	..*.*...	00100000 e-2	7 12	dacă Z=1 dacă Z=0
JP (HL)	PC <- HL	..*.*...	11101001	4	
JP (IX)	PC <- IX	..*.*...	11011101 11101001	8	
JP (IY)	PC <- IY	..*.*...	11111101 11101001	8	
DJNZ	B <- B-1 dacă B=0 continuă altfel PC <- PC+e	..*.*...	00010000	8 13	dacă B=0 dacă B<>0

2.11 SUBRUTINE

Format	Operaţie	Flags	Cod	T	Observaţii
CALL nn	SP <- SP-2 (SP+1) <- PC_h (SP+2) <- PC_l PC <- nn	..*.*...	11001101 -n- -n-	17	c Cond.în F 000 NZ (not 0) 001 Z (=0) 010 NC (C=0) 011 C (C=1) 100 PO (impar) 101 PE (par) 110 P (pozitiv) 111 M (negativ)
CALL c, nn	dacă c fals: continuă dacă c adevărat: SP <- SP-2 (SP+1) <- PC_h (SP+2) <- PC_l PC <- nn	..*.*...	11-c-100 -n- -n-	10 17	dacă c fals dacă c adevărat
RET	PC_l <- (SP) PC_h <-(SP+1) SP <- SP+2	..*.*...	11001101	10	

Format	Operație	Flags	Cod	T	Observații
RET c	dacă c fals: continuă dacă c adevărat: PC_l <- (SP) PC_h <-(SP+1) SP <- SP+2	..*.*...	11-c-000	5 10	dacă c fals dacă c adevărat
RETI	PC_l <- (SP) PC_h <-(SP+1) SP <- SP+2	..*.*...	11101101 01001101	14	ret. din întreruperi mascabile
RETN	PC_l <- (SP) PC_h <-(SP+1) SP <- SP+2 restaurează IFF	..*.*...	11101101 01000101	14	ret. din întreruperi nemascabile
RST p	SP <- SP-2 (SP+1) <- PC_h (SP+2) <- PC_l PC_h <- 0 PC_l <- p	..*.*...	11-t-111	11	t p 000 00h 001 08h 010 10h 011 18h 100 20h 101 28h 110 30h 111 38h

2.12 INPUT ȘI OUTPUT

Format	Operație	Flags	Cod	T	Observații
IN A, (n)	A <-(n)	..*.*...	11011011	11	n în A0 ~ A7 A în A8 ~ A15
IN r, (C)	r <- (C)	!!*!*P0.	11101101 01-r-000	12	C în A0 ~ A7 B în A8 ~ A15 r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
INI	(HL) <- (C) B <- B—1 HL <- HL+1	*!****1*	11101101 10100010	16	C în A0 ~ A7 B în A8 ~ A15 Fz conform B-1
INIR	INI până când B=0	*1****1*	11101101 10110010	21 16	dacă B<>0 dacă B=0

Format	Operaţie	Flags	Cod	T	Observaţii
IND	(HL) <- (C) B <- B—1 HL <- HL—1	*|****1*	11101101 10101010	16	C în A0 ~ A7 B în A8 ~ A15 Fz conform B-1
INDR	IND pînă cînd B=0	*1****1*	11101101 10111010	21 16	dacă B< >0 dacă B=0
OUT (n), A	(n) <- A	..*.*...	11010011 -n-	11	n în A0 ~ A7 A în A8 ~ A15
OUT (C), r	(C) <- r	..*.*...	11101101 01-r-001	12	C în A0 ~ A7 B în A8 ~ A15
OUTI	(C) <- (HL) B <- B—1 HL <- HL+1	*|****1*	11101101 10100011	16	C în A0 ~ A7 B în A8 ~ A15
OTIR	OUTI pînă cînd B=0	*1****1*	11101101 10110011	21 16	dacă B< >0 dacă B=0
OUTD	(C) <- (HL) B <- B—1 HL <- HL—1	*|****1*	11101101 10101011	16	C în A0 ~ A7 B în A8 ~ A15 Fz conform B—1
OTDR	OUTD pînă cînd B=0	*1****1*	11101101 10111011	21 16	dacă B< >0 dacă B=0

Observaţie: Pe lîngă setul de instrucţiuni standard mai există şi altele, avînd codurile în locurile rămase necompletate, din tabelele de dezasamblare. Acestea nu sînt garantate de producător, nu sînt recunoscute de asamblcare, deci utilizarea lor nu este recomandabilă.

Atenţie: Unele instrucţiuni pot fi codificate în mai multe moduri, avînd timpi de execuţie şi număr de bytes diferiţi. Atenţie la modul de codificare al asamblorului folosit, în procesele cu timp critic, calculat (input/output pe bandă magnetică, difuzor, border etc.).

2.13 TABELELE DE DEZASAMBLARE

2.13.1 Tabela 1 (principală), fără prefix

	0	1	2	3	4	5	6	7
00	NOP	LD BC, nn	LD (BC), A	INC BC	INC B	DEC B	LD B, n	RLCA
10	DJNZ d	LD DE, nn	LD (DE), A	INC DE	INC D	DEC D	LD D, n	RLA
20	JR NZ, d	LD HL, nn	LD (nn), HL	INC HL	INC H	DEC H	LD H, n	DAA
30	JR NC, d	LD SP, nn	LD (nn), A	INC SP	INC (HL)	DEC (HL)	LD (HL), n	SCF
40	LD B, B	LD B, C	LD B, D	LD B, E	LD B, H	LD B, L	LD B, (HL)	LD B, A
50	LD D, B	LD D, C	LD D, D	LD D, E	LD D, H	LD D, L	LD D, (HL)	LD D, A
60	LD H, B	LD H, C	LD H, D	LD H, E	LD H, H	LD H, L	LD H, (HL)	LD H, A
70	LD (HL), B	LD (HL), C	LD (HL), D	LD (HL), E	LD (HL), H	LD (HL), L	HALT	LD (HL), A
80	ADD A, B	ADD A, C	ADD A, D	ADD A, E	ADD A, H	ADD A, L	ADD A, (HL)	ADD A, A
90	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A
A0	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A
B0	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A
C0	RET NZ	POP BC	JP NZ, nn	JP nn	CALL NZ, nn	PUSH BC	ADD A, n	RST 00h
D0	RET NC	POP DE	JP NC, nn	OUT (n), A	CALL NC, nn	PUSH DE	SUB n	RST 10h
E0	RET PO	POP HL	JP PO, nn	EX (SP), HL	CALL PO, nn	PUSH HL	AND n	RST 20h
F0	RET P	POP AF	JP P, nn	DI	CALL P, nn	PUSH AF	OR n	RST 30h

	8	9	A	B	C	D	E	F
00	EX AF, AF'	ADD HL, BC	LD A, (BC)	DEC BC	INC C	DEC C	LD C, n	RRCA
10	JR d	ADD HL, DE	LD A, (DE)	DEC DE	INC E	DEC E	LD E, n	RRA
20	JR Z, d	ADD HL, HL	LD HL, (nn)	DEC HL	INC L	DEC L	LD L, n	CPL
30	JR C, d	ADD HL, SP	LD A, (nn)	DEC SP	INC A	DEC A	LD A, n	CCF
40	LD C, B	LD C, C	LD C, D	LD C, E	LD C, H	LD C, L	LD C, (HL)	LD C, A
50	LD E,	LD E,	LD E,	LD E,	LD E,	LD E,	LD E,	LD E,
60	LD L, B	LD L, C	LD L, D	LD L, E	LD L, H	LD L, L	LD L, (HL)	LD L, A
70	LD A, B	LD A, C	LD A, D	LD A, E	LD A, H	LD A, L	LD A, (HL)	LD A, A
80	ADC A, B	ADC A, C	ADC A, D	ADC A, E	ADC A, H	ADC A, L	ADC A, (HL)	ADC A, A
90	SBC A, B	SBC A, C	SBC A, D	SBC A, E	SBC A, H	SBC A, L	SBC A, (HL)	SBC A, A
A0	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B0	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C0	RET Z	RET	JP Z, nn	prefix	CALL Z, nn	CALL nn	ADC A, nn	RST 8h
D0	RET C	EXX	JP C, nn	IN A, (n)	CALL C, nn	prefix	SBC A, n	RST 18h
E0	RET PE	JP (HL)	JP PE, nn	EX DE, HL	CALL PE, nn	prefix	XOR n	RST 28h
F0	RET M	LD SP, (HL)	JP M, nn	EI	CALL PE, nn	prefix	CP n	RST 38h

2.13.2 Tabela 2; prefix: CB

	0	1	2	3	4	5	6	7
00	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A
10	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A
20	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A
30	-	-	-	-	-	-	-	-
40	BIT 0, B	BIT 0, C	BIT 0, D	BIT 0, E	BIT 0, H	BIT 0, L	BIT 0, (HL)	BIT 0, A
50	BIT 2, B	BIT 2, C	BIT 2, D	BIT 2, E	BIT 2, H	BIT 2, L	BIT 2, (HL)	BIT 2, A
60	BIT 4, B	BIT 4, C	BIT 4, D	BIT 4, E	BIT 4, H	BIT 4, L	BIT 4, (HL)	BIT 4, A
70	BIT 6, B	BIT 6, C	BIT 6, D	BIT 6, E	BIT 6, H	BIT 6, L	BIT 6, (HL)	BIT 6, A
80	RES 0, B	RES 0, C	RES 0, D	RES 0, E	RES 0, H	RES 0, L	RES 0, (HL)	RES 0, A
90	RES 2, B	RES 2, C	RES 2, D	RES 2, E	RES 2, H	RES 2, L	RES 2, (HL)	RES 2, A
A0	RES 4, B	RES 4, C	RES 4, D	RES 4, E	RES 4, H	RES 4, L	RES 4, (HL)	RES 4, A
B0	RES 6, B	RES 6, C	RES 6, D	RES 6, E	RES 6, H	RES 6, L	RES 6, (HL)	RES 6, A
C0	SET 0, B	SET 0, C	SET 0, D	SET 0, E	SET 0, H	SET 0, L	SET 0, (HL)	SET 0, A
D0	SET 2, B	SET 2, C	SET 2, D	SET 2, E	SET 2, H	SET 2, L	SET 2, (HL)	SET 2, A
E0	SET 4, B	SET 4, C	SET 4, D	SET 4, E	SET 4, H	SET 4, L	SET 4, (HL)	SET 4, A
F0	SET 6, B	SET 6, C	SET 6, D	SET 6, E	SET 6, H	SET 6, L	SET 6, (HL)	SET 6, A

	8	9	A	B	C	D	E	F
00	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
10	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
20	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
30	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
40	BIT 1, B	BIT 1, C	BIT 1, D	BIT 1, E	BIT 1, H	BIT 1, L	BIT 1, (HL)	BIT 1, A
50	BIT 3, B	BIT 3, C	BIT 3, D	BIT 3, E	BIT 3, H	BIT 3, L	BIT 3, (HL)	BIT 3, A
60	BIT 5, B	BIT 5, C	BIT 5, D	BIT 5, E	BIT 5, H	BIT 5, L	BIT 5, (HL)	BIT 5, A
70	BIT 7, B	BIT 7, C	BIT 7, D	BIT 7, E	BIT 7, H	BIT 7, L	BIT 7, (HL)	BIT 7, A
80	RES 1, B	RES 1, C	RES 1, D	RES 1, E	RES 1, H	RES 1, L	RES 1, (HL)	RES 1, A
90	RES 3, B	RES 3, C	RES 3, D	RES 3, E	RES 3, H	RES 3, L	RES 3, (HL)	RES 3, A
A0	RES 5, B	RES 5, C	RES 5, D	RES 5, E	RES 5, H	RES 5, L	RES 5, (HL)	RES 5, A
B0	RES 7, B	RES 7, C	RES 7, D	RES 7, E	RES 7, H	RES 7, L	RES 7, (HL)	RES 7, A
C0	SET 1, B	SET 1, C	SET 1, D	SET 1, E	SET 1, H	SET 1, L	SET 1, (HL)	SET 1, A
D0	SET 3, B	SET 3, C	SET 3, D	SET 3, E	SET 3, H	SET 3, L	SET 3, (HL)	SET 3, A
E0	SET 5, B	SET 5, C	SET 5, D	SET 5, E	SET 5, H	SET 5, L	SET 5, (HL)	SET 5, A
F0	SET 7, B	SET 7, C	SET 7, D	SET 7, E	SET 7, H	SET 7, L	SET 7, (HL)	SET 7, A

2.13.3 Tabela 3; prefix ED

	0	1	2	3	4	5	6	7
00	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-
30	-	-	-	-	-	-	-	-
40	IN B, (C)	OUT (C), B	SBC HL, BC	LD (nn), BC	NEG	RETN	IM 0	LD I, A
50	IN D, (C)	OUT (C), D	SBC HL, DE	LD (nn), DE	-	-	IM 1	LD A, I
60	IN H, (C)	OUT (C), H	SBC HL, HL	LD (nn), HL	-	-	-	RRD
70	-	-	SBC HL, SP	LD (nn), SP	-	-	-	-
80	-	-	-	-	-	-	-	-
90	-	-	-	-	-	-	-	-
A0	LDI	CPI	INI	OUTI	-	-	-	-
B0	LDIR	CPIR	INIR	OTIR		-	-	-
C0	-	-	-	-	-	-	-	-
D0	-	-	-	-	-	-	-	-
E0	-	-	-	-	-	-	-	-
F0	-	-	-	-	-	-	-	-

	8	9	A	B	C	D	E	F
00	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-
20	-	-	-	-	-	-	-	-
30	-	-	-	-	-	-	-	-
40	IN C, (C)	OUT (C), C	ADC HL, BC	LD BC, (nn)	-	RETI	-	LD R, A
50	IN E, (C)	OUT (C), E	ADC HL, DE	LD DE, (nn)	-	-	IM 2	LD A, R
60	IN L, (C)	OUT (C), L	ADC HL, HL	LD HL, (nn)	-	-	-	RLD
70	IN A, (C)	OUT (C), A	ADC HL, SP	LD SP, (nn)	-	-	-	-
80	-	-	-	-	-	-	-	-
90	-	-	-	-	-	-	-	-
A0	LDD	CPD	IND	OUTD	-	-	-	-
B0	LDDR	CPDR	INDR	OTDR	-	-	-	-
C0	-	-	-	-	-	-	-	-
D0	-	-	-	-	-	-	-	-
E0	-	-	-	-	-	-	-	-
F0	-	-	-	-	-	-	-	-

2.13.4Tabela 4: prefix DD/FD pentru registrul IX/IY

Cod	Instrucțiune	Cod	Instrucțiune
09	ADD I_{xy}, BC	CBd06	RLC (I_{xy}+d)
19	ADD I_{xy}, DE	CBd0E	RRC (I_{xy}+d)
21	LD I_{xy}, nn	CBd16	RL (I_{xy}+d)
22	LD (nn), I_{xy}	CBd1E	RR (I_{xy}+d)
23	INC I_{xy}	CBd26	SLA (I_{xy}+d)
29	ADD I_{xy}, I_{xy}	CBd2E	SRA (I_{xy}+d)
2A	LD I_{xy}, (nn)	CBd3E	SRL (I_{xy}+d)
2B	DEC I_{xy}	CBd46	BIT 0, (I_{xy}+d)

34	INC (I_{xy}+d)	CBd4E	BIT 1, (I_{xy}+d)
35	DEC (I_{xy}+d)	CBd56	BIT 2, (I_{xy}+d)
36	LD (I_{xy}+d),n	CBd5E	BIT 3, (I_{xy}+d)
39	ADD I_{xy}, SP	CBd66	BIT 4, (I_{xy}+d)
46	LD B, (I_{xy}+d)	CBd6E	BIT 5, (I_{xy}+d)
4E	LD C, (I_{xy}+d)	CBd76	BIT 6, (I_{xy}+d)
56	LD D, (I_{xy}+d)	CBd7E	BIT 7, (I_{xy}+d)
5E	LD E, (I_{xy}+d)	CBd86	RES 0, (I_{xy}+d)
66	LD H, (I_{xy}+d)	CBd8E	RES 1, (I_{xy}+d)
6E	LD L, (I_{xy}+d)	CBd96	RES 2, (I_{xy}+d)
70	LD (I_{xy}+d), B	CBd9E	RES 3, (I_{xy}+d)
71	LD (I_{xy}+d), C	CBdA6	RES 4, (I_{xy}+d)
72	LD (I_{xy}+d), D	CBdAE	RES 5, (I_{xy}+d)
73	LD (I_{xy}+), E	CBdB6	RES 6, (I_{xy}+d)
74	LD (I_{xy}+d), H	CBdBE	RES 7, (I_{xy}+d)
75	LD (I_{xy}+d), L	CBdC6	SET 0, (I_{xy}+d)
77	LD (I_{xy}+d), A	CBdCE	SET 1, (I_{xy}+d)
7E	LD A, (I_{xy}+d)	CBdD6	SET 2, (I_{xy}+d)
86	ADD A, (I_{xy}+d)	CBdDE	SET 3, (I_{xy}+d)
8E	ADC A, (I_{xy}+d)	CBdE6	SET 4, (I_{xy}+d)
9B	SUB (I_{xy}+d)	CBdEE	SET 5, (I_{xy}+d)
9E	SBC A, (I_{xy}+d)	CBdF6	SET 6, (I_{xy}+d)
A6	AND (I_{xy}+d)	CBdFE	SET 7, (I_{xy}+d)
AE	XOR (I_{xy}+d)	E1	POP I_{xy}
B6	OR (I_{xy}+d)	E3	EX (SP), I_{xy}
BE	CP (I_{xy}+d)	E5	PUSH I_{xy}
		E9	JP (I_{xy})
		F9	LD SP, I_{xy}

2.13.5 Tabele de conversie: binar/zecimal/hexazecimal

Hex.	Binar	Hex.	Binar
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
20	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
30	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
40	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
50	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
60	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
70	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
80	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
90	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A0	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B0	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C0	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D0	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E0	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F0	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

2.14 ASAMBLORUL GENS*

2.14.1 Instalarea

Asamblorul GENS este cel mai puternic asamblor Z80 pentru calculatoarele compatibile SPECTRUM, avînd următoarele caracteristici:

- este relocabil, putînd fi încărcat la orice adresă în memorie
- editorul este puternic şi eficient
- are instrucţiuni macro

Pentru încărcare şi lansare în execuţie se foloseşte secvenţa:

LOAD "" CODE *număr* **: RANDOMIZE USR** *număr*

unde număr este adresa de încărcare care trebuie sa fie cu aproxmativ 10 kbytes mai mică decît ultima adresă fizică.

Apeluri ulterioare: număr+56: iniţializare totală
număr+61: iniţializare cu păstrarea textului-sursă

Taste funcţionale:

E	Enter:	Carriage return
CS1	Caps shift & 1:	ieşire din comanda-editor
CS0	Delete =Caps shift & 0:	şterge un caracter
CS8	Cursor-> =Caps shift & 8:	avans tab
CS5	Cursor<- =Caps shift & 5:	şterge toată linia

*GENS3M21 este produs înregistrat al firmei HISOFT

2.14.2 Editorul

Promt-ul > aşteaptă o comandă de tipul:

comandă număr-1, număr-2 , şir-1 , şir-2

unde *număr-1*, *număr-2* sînt numere întregi între 1 şi 32767; *şir-1*, *şir-2* sînt şiruri de caractere de lungime între 0 şi 20. Blank-ul nu este semnificativ decît în şiruri; , este separator.

GENERAREA TEXTULUI-SURSĂ

I *număr-1*, *număr-2*

Introduce numărătoarea automată a liniilor.
Valori implicite: număr-1 = 10, număr-2 = 10.

O linie text-sursă este de forma:

număr-linie etichetă instrucţiune operanzi ; comentarii

La fiecare structură, cu excepţia primelor 2, se ajunge cu tab-ul; la terminarea unei linii se apasă E.
Orice linie nouă introdusă, cu comanda I sau direct după prompt-ul > se suprapune peste textul anterior astfel încît liniile anterioare cu acelaşi număr de linie se şterg.

LISTARE

L *număr-1*, *număr-2*

Listează liniile textului-sursă avînd numerele între *număr-1* şi *număr-2*; orice tastă efectuează scroll, cu excepţia CS1.
Valori implicite: *număr-1* = 1, *număr-2* = 32767

K *număr*

Setează numărul de linii de listat înainte de scroll.
Valori implicite: *număr* = 16

EDITARE

D *număr-1*, *număr-2*

Şterge liniile textului-sursă, avînd numerele între *număr-1* şi *număr-2*.
Valori implicite: fără
Dacă *număr-1* > *număr-2* sau lipsesc argumente comanda este ignorată..

M *număr-1*, *număr-2*

Introduce linia număr-1 la linia *număr-2*;linia *număr-1* rămîne în text; linia *număr-2* anterioară se şterge.
Valori implicite: fără

N *număr-1*, *număr-2*

Renumerotează textul-sursă începînd cu număr-1 şi cu pasul număr-2
Valori implicite: fără

F *număr-1, număr-2, şir-1, şir-2*
şir-1 este căutat în liniile cu numerele între *număr-1* şi *număr-2*; dacă este găsit atunci se intră în editorul de linie (comanda E) cu cursorul pe primul caracter al *şir-1*; *şir-2* substituie *şir-1*.
Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără

E *număr*
Editor de linie.
Valori implicite: fără
Dacă *număr* lipseşte, comanda este ignorată.
Sub-comenzi:

CS0	cursor stînga
CS8	avans tab
E	terminarea editării, cu modificări
C	supra-scriere, începînd de la poziţia curentă a cursorului
F	caută următorul *şir-1* din comanda F
I	inserează caractere la poziţia curentă a cursorului
K	şterge un caracter la poziţia cursorului
L	listează restul liniei în curs de editare
Q	terminarea editarii, modificările sînt anulate
R	anulează modificările
S	înlocuieşte *şir-1* din comanda F
X	avansează cursorul la sfîrşitul liniei şi introduce sub-comanda I
Z	şterge toate caracterele de la cursor la sfîrşitul liniei

COMENZI DE BANDĂ MAGNETICĂ

P *număr-1, număr-2, şir*
Salvează pe bandă magnetică liniile din textul-sursă între *număr-1* şi *număr-2*, într-un fişier cu numele *şir*.
Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără.

G , , *şir*
Încarcă de pe bandă magnetică fişierul (text-sursă) şir la sfîrşitul textului-sursă şi liniile sînt renumerotate.
Valori implicite: dacă şir lipseşte se încarcă primul fişier

T *număr-1, număr-2, şir*
Salvează liniile, avînd numărul între *număr-1* şi *număr-2* cu numele *şir* în formatul utilizabil de directiva de asamblare *F.

COMENZI DE UZ GENERAL

B
Retur în BASIC Spectrum.

C
Interoghează utilizatorul asupra buffer-ului utilizat de directiva de asamblare *F (Include buffer) şi a celui utilizat de instrucţiunile Macro (Macro buffer). Bufferele

sînt setate la dimensiunea minimă 256. Dacă dimensiunile nu sînt date comanda este abandonată.

S , , *d*

Schimbă separatorul operanzilor din , în *d*; *d* nu poate fi blank.

V

Afişează bufferul.

X

Afişează în zecimal adresele de început şi sfîrşit ale textului-sursă.

W *număr-1*, *număr-2*

Tipăreşte la imprimantă liniile textului-sursă, avînd numerele între *număr-1* şi *număr-2*.
Valori implicite: dacă *număr-1* şi *număr-2* lipsesc atunci se tipăreşte tot textul.

ASAMBLAREA ŞI RULAREA PROGRAMELOR

A

Asamblează textul-sursă. Comanda A cere dimensiunea tabelei de simboluri (table size) în zecimal (la apăsarea Enter se consideră o valoare implicită funcţie de lungimea textului-sursă) şi opţiunile de asamblare; număr în zecimal obţinut prin adunarea numerelor opţiunilor selectate:

1 afişează tabela de simboluri
2 nu generează cod-obiect
4 nu listează asamblarea
8 listează codul asamblat la imprimantă
16 plasează codul-obiect după tabela de simboluri, codul-obiect este destinat, însă, să ruleze la adresele ORG
32 nu verifică adresele de asamblare ORG

Directivele de asamblare sînt plasate ca linii-program separate de numărul de linie printr-un blank. Acestea sînt:

*C+/- activează/inhibă listarea codului generat în linia asamblată
*D+/- adresele, afişate după aceasta directivă,sînt în zecimal/ hexazecimal
*E separator de module, creează 3 linii vide la listare
*F *şir* asamblează fişierele de pe bandă magnetică salvate cu comanda T; *şir* este opţional (directivă utilă pentru programe foarte mari)
*L+/- activează/inhibă listarea începînd de la această linie
*H *şir* efectuează *E şi afişează *şir* ca titlu
*S opreşte listarea la această linie, listarea se reia prin apăsarea oricărei taste

O linie asamblată, la listare, are forma:

adresă *cod* *număr-linie* *etichetă* *instrucţiune* *operanzi*

Dacă tabela de simboluri este prea mică se generează mesajul: **Out of table space !** şi asamblarea este abandonată.
Dacă textul-sursă este prea mare se generează mesajul: **Bad memory !** şi asamblarea este abandonată.

R

Rulează codul asamblat avînd adresa de start, data de directiva de asamblare ENT. Cu ajutorul instrucţiunii RET se poate reintra în asamblor, la sfîrşitul rulării, dacă stiva microprocesorului este restaurată la ieşire.

2.14.3 DIRECTIVE DE ASAMBLARE

EXPRESII

O expresie este o listă de termeni separaţi de operatori, unde:

termen:	constantă zecimală	
	constantă hexazecimală precedata de #	
	constantă binară precedată de %	
	caracter încadrat între ghilimele	
	etichetă, maxim 6 caractere, care începe cu o literă	
	$, semnifică adresa curentă	
operator:	+	adunare
	-	scădere
	&	operaţia AND
	@	operaţia OR
	*	înmulţire între întregi
	/	împărţire între întregi
	?	operaţia MODULO: x?y=x-(x/y)*y

Din numere se consideră partea MODULO 65535 (#10000); expresiile sînt evaluate de la stînga la dreapta, fără priorităţi; o expresie în paranteză, ca operand, semnifică o adresă..

DIRECTIVE

DEFB *expresie-1, ...*

Fiecare expresie,evaluată la un byte, este stocată la adresa curentă a DEFB.

DEFS *expresie*

Rezervă, începînd de la adresa curentă, un număr de bytes egal cu valoarea *expresie*.

DEFS *"şir"*

Stochează în memorie, începînd de la adresa curentă, codurile ASCII ale caracterelor care formează şirul (în ordinea în care apar în şir).

DEFW *expresie-1,...*

Fiecare expresie, evaluată la 2 bytes, este stocată la adresa curentă a DEFW.

ELSE

Dacă liniile de dinainte de ELSE nu se asamblează (ca urmare a IF) atunci liniile de după ELSE vor fi asamblate şi reciproc; se utilizează numai în structurile IF ... ELSE ... END, unde este opţional.

END
Marchează sfîrşitul structurii IF ... ELSE ... END.

ENT *expresie*
Adresa de start, pentru comanda R, are valoarea *expresie*.

eticheta **EQU** *eticheta*
Expresia, de tip număr, poate fi înlocuită cu eticheta.

IF *expresie*
Dacă expresie=0 liniile care urmează după IF, pîna la ELSE, END sau sfîrşitul textului, nu sînt asamblate.

ORG *expresie*
Următoarele linii de text-sursă vor fi asamblate începînd de la adresa expresie, pînă la întîlnirea unei noi ORG sau pînă la sfîrşitul textului-sursă.

; *comentariu*
Caracterele care urmează după ; sînt ignorate la asamblare (se consideră comentariu).

Observaţie: IF/ELSE/END se utilizează în mod normal în forma:

IF *expresie*
text-sursa-1
ELSE
text-sursa-2
END

MACRO-INSTRUCŢIUNI

Definirea macro-instrucţiunilor:

nume **MAC**
...
... (corpul definiţiei)
...
ENDM

O definiţie macro poate avea pînă la 32 de parametri, notaţi în definiţie, cu simbolurile **=0**, ..., **=31**.
Apelul unei macro-instrucţiuni se face astfel:

nume parametru-0, ...

parametrii putînd fi utilizaţi şi în expresii.

2.14.4Codul erorilor generate

1 eroare în linie
2 mnemonic nerecunoscut
3 declaraţie eronată
4 simbol definit de mai multe ori
5 caracter ilegal în linia respectivă
6 operand ilegal în linia respectivă
7 simbolul utilizat este cuvînt rezervat (mnemonic, nume-registru sau directivă-asamblor)
8 regiştri nepotriviţi
9 prea mulţi regiştri
10 o expresie care ar fi trebuit să fie evaluată la un byte are o valoare mai mare
11 instrucţiunea JP (IX+d) sau JP (IY+d) nu există
12 eroare în directiva asamblor
13 simbol în EQU care nu a fost definit încă
14 împărţire cu 0
15 depăşire la înmulţire
16 lipseşte definiţia macro-instrucţiunii
17 numele nu este identificator al unei macro-instrucţiuni
18 lipseşte un apel macro
19 lipseşte declaraţia condiţională

2.15 DEZASAMBLORUL MONS*

2.15.1 Instalarea

Dezasamblorul MONS este destinat să lucreze în tandem cu asamblorul GENS astfel încît este relocabil.
Pentru încărcare şi lansare în execuţie se foloseşte secvenţa:

LOAD "" CODE *număr* **: RANDOMIZE USR** *număr*

unde număr este adresa de încărcare. Necesarul de memorie pentru MONS este de aproximativ 6Kbytes.
Apel ulterior: **USR** *număr*+**24**
Ecranul principal are structura următoare:

*MONS3M21 este produs înregistrat al firmei HISOFT

adresa-curentă		cod	instrucţiune			
PC	număr-1	nr-1	...	nr-7	(7 numere ,	de la adr. PC)
SP	număr-2	.		.		
IY	număr-3	.		.		
IX	număr-4	.		.		
HL	număr-5					
DE	număr-6					
BC	număr-7	nr-42	...	nr-49	(7 numere,	de la adr. BC)
AF	număr-8	ZS H P/V NC	(sint afişate			
IR	număr-9		numai			
			fanioanele			
			setate)			
adresa-1		n-1	adresa-9	n-9	adresa-17	n-17
.						
.			> adr. curentă	n13<		
.						
adresa-8		n-8	adresa-16	n-16	adresa-24	n-24
>						

Prompt-ul inferior semnifică aşteptarea unei comenzi.

2.15.2 Comenzi

Enter
Incrementează adresa curentă cu 1.

CS şi 5 (cursor ^)
Decrementează adresa curentă cu 8.

CS şi 7 (cursor ->)
Decrementează adresa curentă cu 1.

CS şi 8 (cursor <-)
Incrementează adresa curentă cu 8.

SS şi 3
Comută zecimal/hexazecimal modul de afişare al adreselor şi regiştrilor PC, ..., BC.

SS şi 4
Comută ecran principal/pagina de cod dezasamblat în mod scroll.

SS şi P

Tipăreşte un bloc de memorie, la imprimantă, de la adresa curentă şi caracterele corespunzătoare, în standardul ASCII (pentru codurile mai mari decît 127 bitul 7 este resetat apoi pentru codurile mai mici decît 32 se afişează .). CS şi 5 efectuează reîntoarcere la ecranul principal; CS şi 1 retur în BASIC Spectrum; orice alta tastă efectuează scroll.

SS şi T

Introduce un punct de control după instrucţiunea curentă şi continuă execuţia. Punctele de control introduse de comanda W se pierd. PC şi adresa curentă trebuie să coincidă.

SS şi Z

Realizează rularea programului instrucţiune cu instrucţiune; PC şi adresa curentă trebuie să coincidă.

,

Adresa curentă devine egală cu numărul de pe stivă.

G

Caută şir: fiecare byte hexazecimal este dat pe rînd prin tastarea Enter; şirul se termină cu orice caracter care nu este cifră hexazecimală.

H

Converteşte un număr decimal în hexazecimal; numărul se termină cu orice caracter care nu este număr hexazecimal.

I

Copiază un bloc de memorie; **First:** adresa de început, **Last:** adresa de sfîrşit, **To:** adresa destinaţie. Se copiază inclusiv blocurile care rescriu peste vechiul bloc.

J

Salt şi execuţie la adresa specificată; dacă se doreşte întoarcerea în dezasamblor se introduce un punct de control (comanda W). Comanda J nu păstrează conţinutul actual al regiştrilor.

K

Continuă execuţia de la adresa curentă, nu afectează conţinutul regiştrilor, dacă se doreşte reîntoarcerea în dezasamblor se introduce un punct de control (comanda W).

L

Listează un bloc de memorie, de la adresa curentă, şi caracterele corespunzătoare, în standardul ASCII (pentru codurile mai mari decît 127 bitul 7 este resetat apoi pentru codurile mai mici decît 32 se afişează .). CS şi 5 efectuează reîntoarcere la ecranul principal; CS şi 1 retur în BASIC Spectrum; orice alta tastă efectuează scroll.

M

Modifică adresa curentă conform celei furnizate de utilizator în hexazecimal.

N

Caută locul următor în care se găseste şirul specificat de ultima comandă G.

O

Byte-ul de la adresa curentă se consideră o adresare relativă (în instrucţiunea JR) şi adresa curentă se modifică în conformitate cu acesta (reîntoarcere cu comanda U).

P

Umple un bloc de memorie de la adresa **First:** la adresa **Last:** cu byte-ul **With:**.

Q

Comută pe setul alternat de regiştri.

T

Dezasamblează un bloc de memorie de la adresa **First:** la adresa **Last:** la imprimantă dacă se introduce y sau pe ecran fără scroll. **Text:** dezasamblează text, Enter, simplu, anulează comanda.

U

Reîntoarcere la adresa curentă (iniţială) după comanda O.

V

Reîntoarcere la adresa curentă (iniţială) după comanda X.

W

Introduce un punct de control (oprire) la adresa curentă, aceasta fiind de forma **CALL** *număr* (3 bytes). După utilizarea acestui punct de control cei 3 bytes sînt refăcuţi. Se pot utiliza oricîte puncte de control, fiecare necesitînd 5 bytes spaţiu liber în memorie după MONS.

X

Cei 2 bytes de la adresa curentă se consideră o adresa de salt (instrucţiunile CALL şi JP) şi adresa curentă se modifică în conformitate cu aceştia (reîntoarcere cu comanda V).

Y

Permite introducerea unui şir alfanumeric care este introdus începînd de la adresa curentă a memoriei. Şirul se termina cu CS şi 5. Delete permite ştergerea caracterelor. Adresa curentă devine prima adresă după şir.

Modificarea memoriei

Conţinutul adresei curente poate fi modificat direct prin introducerea unui număr hexazecimal. Dupa introducerea numărului adresa curentă se incrementează. Numărul nu trebuie să se termine cu un simbol care reprezintă o comandă.

Modificarea regiştrilor

. deplaseaza cursorul registru (ecranul principal: stînga sus) prin permutare circulară. *număr* . introduce *număr* în registrul curent.

3 BASIC

3.1 BASIC SPECTRUM

3.1.1 Constante, variabile, expresii

Constantă numerică

Format standard:

semn cifră. cifră...cifră **E** *semn cifră cifră*

semn este + (opţional) sau -. *cifră* este un element din mulţimea {0, ..., 9}. Cifrele din faţa punctului zecimal şi exponentul pot lipsi dacă sînt egale cu 0. Numerele sînt afişate în formatul standard.

Constante-şir

Format:

şir

Unde *sir* este o serie de caractere afişabile (cu cursorul L, C, E, G cu sau fără SS).

Variabile numerice

Variabilele care înlocuiesc o valoare numerică unică pot fi formate din mai multe litere şi cifre; trebuie să înceapă cu o literă; nu se face distincţia între literele mari şi mici.

Variabilele care înlocuiesc tablouri-numerice pot fi formate dintr-o singură literă; nu se face distincţia între literele mari şi mici.

Variabile-şir

Variabilele care înlocuiesc şiruri sau tablouri-şir pot fi formate dintr-o singură literă; nu se face distincţia între litere mari şi mici.

Observaţie: numele unei variabile numerice, şir sau contor pot coincide fără a exista confuzii/erori.

Expresii

Expresiile sînt liste de constante, variabile, funcţii BASIC şi funcţii definite de programator, separate de operatori şi paranteze (,).
Expresiile sînt evaluate de la stînga la dreapta, conform ordinii de prioritate a operatorilor.
Parantezele (,) se utilizează pentru a semnaliza inversarea ordinii de aplicare a priorităţii operatorilor. Modul de evaluare a unei expresii în ordinea descrescatoare a priorităţilor este:
a) se evaluează subexpresiile din paranteze
b) se evaluează funcţiile
c) se evaluează subexpresiile aritmetice; ordinea de prioritate este:
   ^ ridicare la putere (priorite maximă)
   *, / înmulţire, împărţire
   +, - adunare, scădere (prioritate minimă)
d) se evaluează subexpresiile operaţionale; operatorii operaţionali:
   >, =, <, <=, >=, <> (au aceeaşi ordine de prioritate)
e) se evaluează subexpresiile logice; ordine de prioritate:
   NOT (prioritate maximă)
   AND
   OR (prioritate minimă)
Expresiile sînt de 2 tipuri după rezultatul evaluării:
   expresii-numerice generează numere
   epresii-şir generează şiruri

3.1.2 Cuvinte cheie BASIC Spectrum

Cuvintele cheie BASIC Spectrum sînt de 3 tipuri: comenzi, funcţii şi operatori (cu destinaţii diverse).

ABS

Format:
   ABS *constantă/variabilă-numerică* sau
   ABS (*expresie-numerică*)
ABS este funcţie şi returnează valoarea absolută (modulul) numărului.

ACS

Format:
   ACS *constantă/variabilă-numerică* sau
   ACS (*expresie-numerică*)
ACS este funcţie şi returnează arccosinusul numărului (între -1 şi 1)în radiani.

AND

Format:
   condiţie-1 **AND** *condiţie-2* (a) sau
   expresie-numerică-1 **AND** *expresie-numerică-2* (b) sau
   expresie-şir **AND** *expresie-numerică* (c)
unde condiţie-1/2 sînt expresii care prin evaluare (a operatorilor operaţionali sau logici) pot fi false sau adevărate. AND este funcţie şi returnează:
(a) adevărat (funcţie logică) dacă *condiţie-1/2* sînt amîndouă adevărate; altfel fals

(b) *valoare-numerică-1* a expresiei dacă *valoare-numerică-2* a *expresiei-2* este nenulă
(c) *valoare-şir* a *expresiei-şir* dacă *valoare-numerică* a *expresiei-numerice* este nenulă

ASN

Format:

ASN *constantă/variabilă-numerică* sau
ASN (*expresie-numerică*)

ASN este funcţie şi returnează arcsinusul numărului (între -1 şi 1) în radiani.

AT

Format:

AT *linie, coloană*

AT este operator şi stabileşte poziţia de afişare în comenzile INPUT şi PRINT. *linie* este numărul liniei, între 0 şi 21, de sus în jos; *coloană* este numărul coloanei, între 0 şi 31, de la stînga la dreapta. *linie* şi *coloană* pot fi constante, variabile sau expresii numerice, rotunjite dacă este cazul la numărul întreg cel mai apropiat.

ATN

Format:

ATN *constantă/variabilă-numerică* sau
ATN (*expresie-numerică*)

ATN este funcţie şi returnează arctangentul numărului în radiani.

ATTR

Format:

ATTR (*linie,coloană*)

ATTR este funcţie şi returnează valoarea atributului aflat în poziţia (*linie, coloană*); *linie* este numărul liniei, între 0 şi 21, de sus în jos; *coloană* este numărul coloanei, între 0 şi 31, de la stînga la dreapta. *linie* şi *coloană* pot fi constante, variabile sau expresii numerice, rotunjite dacă este cazul la numărul întreg cel mai apropiat. Atributul unui caracter este:

Cod culoare:		Cod BRIGHT	Cod FLASH
0 negru	4 verde	0 normal	0 normal
1 roşu	5 cyan	1 strălucitor	1 clipitor
2 albastru	6 galben		
3 magenta	7 alb		

BEEP

Format:

BEEP *expresie-numerică-1, expresie-numerică-2*

BEEP este comandă şi emite un sunet la difuzor. *expresie-numerică-1* reprezintă durata în secunde, între 0 şi 10; *expresie-numerică-2* reprezintă înălţimea notei, între -60 şi 69. Diferenţa de o unitate între înălţimile a 2 note reprezintă o diferenţă de un semiton. Nota DO, inferior, din cheia SOL are înălţimea 0.

BIN

Format:

BIN *număr-binar*

BIN este funcţie şi returnează valoarea numărului binar în zecimal (între 0 şi 65535 zecimal = număr binar cu maxim 16 cifre binare, 0 sau 1).

BORDER

Format:

BORDER *expresie-numerică-pozitivă*

BORDER este comandă şi schimbă culoarea marginii conform valorii expresiei numerice, între 0 şi 7, eventual aceasta fiind rotunjită la întregul cel mai apropiat.
Cod culoare:

0 negru	4 verde
1 roşu	5 cyan
2 albastru	6 galben
3 magenta	7 alb

BRIGHT

Format:

BRIGHT *expresie-numerică-pozitivă*

BRIGHT este comandă/operator şi permite comandarea intensităţii luminoase a afişării conform valorii expresiei numerice, eventual aceasta fiind rotunjită la valoarea întreagă cea mai apropiată:

expresie-numerică = 0	intensitate luminoasă normală
expresie-numerică = 1	intensitate luminoasă mărită (strălucitor)
expresie-numerică = 8	opţiunile BRIGHT 0/1 făcute anterior asupra poziţiilor de afişare rămin valabile la afişări ulterioare

CAT

Comandă microdrive.

CHR$

Format:

CHR$ *constantă/variabilă-numerică-pozitivă* sau
CHR$ (*expresie-numerică-pozitivă*)

CHR$ este funcţie şi returnează caracterul al cărui cod este dat de *număr*. Eventual numărul este rotunjit la numărul întreg cel mai apropiat.

CIRCLE

Format:

CIRCLE *listă-opţiuni expresie-numerică-pozitivă-1, expresie-numerică-pozitivă-2, expresie-numerică-pozitivă-3*

unde *listă-opţiuni* este opţională şi este de forma:

opţiune-afişare ; ...

şi *opţiune-afişare* este de tipul: PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE.

CIRCLE este comandă şi trasează pe ecran un cerc aproximativ:

expresie-numerică-poztivă-1 reprezintă coordonata x a centrului; valori extremale de la 0 la 255, de la stînga la dreapta

expresie-numerică-pozitivă-2 reprezintă coordonata y a centrului; valori extremale de la 0 la 175, de jos în sus

expresie-numerică-pozitivă-3 reprezintă raza, valori extremale de la 0 la 87

Expresiile numerice sînt eventual rotunjite la valoarea întreagă cea mai apropiată şi trebuie să aibă valorile astfel încît cercul să se încadreze în ecran. Cercul este trasat din extremitatea stîngă în sens orar.

CLEAR

Format:

CLEAR *expresie-numerică-pozitivă*

CLEAR este o comandă şi realizează următoarele operaţii:

a) şterge toate variabilele BASIC
b) şterge ecranul
c) atribuie variabilei-sistem RAMTOP valoarea *expresiei-numerice* eventual rotunjită la valoarea întreagă cea mai apropiată; expresia numerică poate lipsi caz în care se consideră valoarea implicită 65367.

RAMTOP reprezintă adresa ultimei locații utilizabile de programul BASIC.

CLOSE#

Comandă microdrive.

CLS

Format:

CLS

CLS este comandă și realizează ștergerea ecranului și resetarea atributelor conform culorilor INK și PAPER permanente.

CODE

Format:

CODE *constantă/variabilă-șir* (a)
CODE (*expresie-șir*) (b)
CODE *expresie-numerică-pozitivă-1*, *expresie-numerică-pozitivă-2* (c)

CODE este funcție/operator și realizează:

(a),(b) funcție care returnează codul primului caracter al șirului (codurile caracterelor afișabile)

(c) operator utilizat în comenzile LOAD, SAVE și VERIFY pentru manipularea blocurilor de bytes pe bandă magnetică
expresie-numerică-pozitivă-1 reprezintă adresa primului byte
expresie-numerică-pozitivă-2 reprezintă numărul de bytes

CONTINUE

Format:

CONTINUE

CONTINUE este comandă și este utilizată pentru continuarea rulării programului după o comandă STOP sau BREAK.

COPY

Format:

COPY

COPY este comandă și realizează copia ecranului la imprimantă.

COS

Format:

COS *constantă/variabilă-numerică* sau
COS (*expresie-numerică*)

COS este funcție și returnează valoarea cosinusului numărului considerat unghi măsurat în radiani.

DATA

Format:

DATA *expresie-numerică/șir*,... (a)sau
DATA *literă*( ) (b)sau
DATA *literă*$( ) (c)

DATA este comandă/operator și realizează:

(a)comandă: utilizată pentru generarea datelor citite de comanda READ; pot fi puse oriunde în program, respectînd ordinea lor de succesiune

(b),(c)operator: utilizat de comenzile LOAD, SAVE și VERIFY pentru manipularea tablourilor numerice/șir pe bandă magnetică

Variabilele din DATA trebuie să aibă valori în momentul citirii READ dacă DATA este utilizată ca o comandă.

DEF FN

Format:

DEF FN *literă listă-variabile = expresie-numerică* sau
DEF FN *literă*$ *listă-variabile = expresie-şir*

unde *listă-variabile*, opţională, are formatul:

(*variabilă-numerică/şir* , ...)

DEF FN este comandă şi este utilizată pentru definirea unei funcţii-utilizator numerice sau şir. Funcţia definită depinde de variabilele din listă.

DIM

Format:

DIM *literă* (*dimensiune-1*, ...) sau
DIM *literă* $ (*dimensiune-1* , ...)

unde *dimensiune* este de forma:

expresie-numerică-pozitivă

în care eventualele variabile trebuie să aibă valori în momentul evaluării; expresiile numerice sînt rotunjite la numărul întreg cel mai apropiat; numărul dimensiunilor este nelimitat.

DIM este comandă şi realizează creearea şi iniţializarea variabilelor tablou numerice/şir cu 0/blank-uri.

DRAW

Format:

DRAW *listă-opţiuni expresie-numerică-1, expresie-numerică-2* (a) sau
DRAW *listă-opţiuni expresie-numerică-1, expresie-numerică-2,* (b)
expresie-numerică-3

unde *listă-opţiuni* este opţională şi este de forma:

opţiune-afişare ; ...

şi *opţiune-afişare* este de tipul PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE.

DRAW este comandă şi efectuează pe ecran:

(a) trasarea unei drepte între punctele de coordonate (X0, Y0) şi (X0 + *expresie-numerică-1*, Y0 + *expresie-numerică-2*)

(b) trasarea unui arc de cerc între punctele de coordonate (X0, Y0) şi (X0 + *expresie-numerică-1*, Y0 + *expresie-numerică-2*) a cărui unghi la centru măsurat în radiani este *expresie-numerică-3*; dacă *expresie-numerica-3* > 0 trasarea se face în sens trigonometric, altfel în sens orar

unde (X0, Y0) reprezintă coordonatele ultimului pixel trasat de o comandă PLOT, DRAW, CIRCLE anterioară, (0, 0) iniţial sau după CLS.

x măsurat de la stînga la dreapta, de la 0 la 255

y măsurat de jos în sus, de la 0 la 175

ERASE

Comandă microdrive.

FLASH

Format:

FLASH *expresie-numerică-pozitivă*

FLASH este comandă/operator şi permite comandarea afişarii clipitoare conform valorii expresiei numerice, eventual rotunjită la valoarea întreagă cea mai apropiată.

expresie-numerică = 1: afişare clipitoare
expresie-numerică = 0: afişare normală
expresie-numerică = 8: opţiunile făcute anterior asupra poziţiilor de afişare rămîn nemodificate la afişări ulterioare.

FN

Format:

FN *literă listă-expresii* sau

FN *literă* **$** *listă-expresii*

unde *listă-variabile* are formatul:

(*expresie-numerică/şir* , ...)

FN este funcţie şi returnează valoarea funcţiei definită prin DEFN respectiv. În momentul evaluării variabilelor din *lista-variabile* (DEFN) li se atribuie valorile obţinute în urma evaluării expresiilor, în ordinea din liste.

FOR

Format:

FOR *literă* = *expresie-numerică-1* **TO** *expresie-numerică-2 opţiune-pas*

unde *opţiune-pas* este de tipul STEP.

FOR este comandă şi permite realizarea buclelor cu contor.

Variabila-contor este literă şi este incrementată cu 1, dacă nu a fost specificat alt pas prin *opţiune-pas*; începînd cu *expresie-numerică-1*, atîta timp cît este mai mică sau egală cu *expresie-numerică-2*. Corpul buclei este cuprins între instrucţiunea FOR şi instrucţiunea NEXT corespunzătoare. Sînt permise oricîte bucle imbricate, buclele încrucişate sînt interzise.

FORMAT

Comandă microdrive.

GO SUB

Format:

GO SUB *expresie-numerică-pozitivă*

GO SUB este comandă şi permite saltul la o subrutină care începe la numărul de linie egal cu *expresie-numerică* care eventual este rotunjită la numărul întreg cel mai apropiat. Dacă nu există un astfel de număr atunci programul continuă cu linia avînd numărul imediat următor. Reîntoarcerea în rutina apelantă se execută cu RETURN.

GO TO

Format:

GO TO *expresie-numerică-pozitivă*

GO TO este comandă şi permite saltul la o linie care are numărul egal cu *expresie-numerică* care este eventual rotunjită la numărul întreg cel mai apropiat. Dacă nu există un astfel de număr de linie atunci programul continuă cu linia avînd numărul de linie imediat următor.

IF

Format:

IF *condiţie* **THEN** *listă-instrucţiuni*

unde *condiţie* este o expresie care poate fi adevărată sau falsă şi *listă-instrucţiuni* are formatul:

instrucţiune **:** ...

IF este comandă şi permite execuţia condiţionată a instrucţiunilor:

a) condiţie adevărată: se execută *listă-instrucţiuni*

b) condiţie falsă: nu se execută *listă-instrucţiuni*

listă-instrucţiuni este opţională.

IN

Format:

IN *constantă/variabilă-numerică* sau

IN (*expresie-numerică*)

IN este funcţie şi returnează ca valoare byte-ul citit la portul de intrare a cărui adresă este numărul respectiv.

Porturile tastaturii au adresa de tipul: 256*(255-2n)+254; n: {0,...,7}

0 CS...V	4 0...6
1 A...G	5 P...Y
2 Q...T	6 Enter...H
3 1...5	7 Space...B

INK

Format:

INK *expresie-numerică*

INK este comandă/operator şi permite controlul culorii de scriere, conform *expresiei-numerice* care eventual este rotunjită la valoarea întreagă cea mai apropiată.

Culori:

0 negru	5 cyan
1 albastru	6 galben
2 roşu	7 alb
3 magenta	8 transparent
4 verde	9 contrast

transparent: culoarea INK nu se modifică la afişări ulterioare
contrast: culoarea INK este negru/alb vizibil pe culoarea PAPER

INKEY$

Format:

INKEY$

INKEY$ este funcţie şi returnează ca valoare un şir cu un caracter, dat de tasta care este apăsată în momentul evaluării; dacă nu a fost apăsată nici o tastă se returnează şirul vid; nu se aşteaptă apăsarea unei taste; se detectează combinaţii de taste cu CS şi SS.

INPUT

Format:

INPUT *element-input separator ...*

unde *element-input* este de forma:

element separator ... sau
(*element separator ...*)

a)(,)determină afişarea informaţiilor dintre paranteze
b)element este de forma:
constantă/variabilă-numerică/şir
opţiune-afişare: PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE
LINE *constantă/variabilă-şir*: determină suprimarea ghilimelelor la afişare
AT *linie, coloană* : pentru poziţia de afişare

şi separator este de forma:

; poziţia următoare de afişare este în continuare, este separator pentru opţiunile de afişare
, poziţia următoare de afişare este în coloana 0/16
' poziţia următoare de afişare este în linia următoare

INPUT este comandă şi permite introducerea datelor de la tastatură de către utilizator, în timpul rulării programului.

INT

Format:

INT *constantă/variabilă-numerică* sau
INT (*expresie-numerică*)

INT este funcţie şi returnează partea întreagă a numărului.

INVERSE

Format:

INVERSE *expresie-numerică-pozitivă*

INVERSE este comandă/operator şi permite afişarea în culori inversate (culoarea INK se înlocuieşte cu PAPER şi reciproc):

0 afişare normală
1 afişare inversată

LEN

Format:

LEN *constantă/variabilă-şir* sau
LEN (*expresie-şir*)

LEN este funcţie şi returnează lungimea şirului.

LET

Format:

LET *variabilă-numerică/şir* = *expresie-numerică/şir*

LET este comandă şi permite asignarea variabilelor cu valorile expresiilor.

LINE

Format:

LINE *constantă/variabilă-şir* (a) sau
LINE *expresie-numerică-pozitivă* (b)

LINE este opţiune şi permite:

(a) suprimarea ghilimelelor de încadrare a şirului în comanda INPUT
(b) autorularea programelor, la încărcare (LOAD), la numărul de linie dat de expresie, eventual rotunjită la numărul întreg cel mai apropiat, dacă acestea au fost salvate (SAVE) cu această opţiune; dacă un astfel de număr de linie nu există atunci programul începe cu linia avînd numărul imediat următor

LIST

Format:

LIST *expresie-numerică-pozitivă*

LIST este comandă şi permite listarea programului pe ecran începînd cu numărul de linie dat de *expresie-numerică*, eventual aceasta fiind rotunjită la numărul întreg cel mai apropiat. *Expresia-numerică* este opţională, valoare implicită 0. Poziţionează cursorul de selectare pentru editarea liniei.

LLIST

Format:

LLIST *expresie-numerică-pozitivă*

LLIST este comandă şi permite listarea programului la imprimantă, începînd cu numărul de linie dat de *expresie-numerică*, eventual aceasta fiind rotunjită la numărul întreg cel mai apropiat. *Expresia-numerica* este opţională, valoare implicită 0.

LN

Format:

LN *constantă/variabilă-numerică-pozitivă* sau
LN (*expresie-numerică-pozitivă*)

LN este funcţie şi returnează ca valoare logaritmul natural al numărului.

LOAD

Format:

LOAD "*expresie-şir*" *opţiune-date*

unde *opţiuni-date* sînt opţionale şi sînt de tipul CODE, DATA sau SCREEN$.

LOAD este comandă şi permite încărcarea fişierelor de pe bandă magnetică. *Expresie-şir* este opţională şi dacă lipseşte se încarcă primul fişier de pe bandă magnetică. Dacă *opţiunea-date* lipseşte se încarcă un program BASIC. Dacă parametrii opţiunii CODE lipsesc atunci blocul de bytes se încarcă conform parametrilor salvaţi.

LPRINT

Format:

LPRINT *listă-opţiuni element-imprimare separator ...*

listă-opţiuni este de forma:

opţiune-imprimare ; ...

opţiune-imprimare este de tipul AT, TAB, INVERSE sau OVER

element-imprimare este de forma:

expresie-numerică/şir

separator este:

; poziţia următoare de tipărire este în continuare

, poziţia următoare de tipărire este în coloana 0/16

' poziţia următoare de tipărire este în linia următoare

LPRINT este comandă şi permite tipărirea la imprimantă.

MERGE

Format:

MERGE "*expresie-şir* "

MERGE este comandă şi permite concatenarea programului BASIC din memorie cu unul înregistrat pe bandă magnetică. MERGE încarcă un program de pe bandă magnetică şi îl concatenează cu cel din memorie, intercalînd, în ordine, numerele de linie. Liniile din programul din memorie se şterg dacă există în programul de pe bandă linii cu acelaşi număr de linie. MERGE stopează autorularea programelor.

MOVE

Comandă microdrive.

NEW

Format:

NEW

NEW este comandă care realizează restart-ul soft al calculatorului.

NEXT

Format:

NEXT *literă*

NEXT este comandă şi închide bucla deschisă de comanda FOR, căreia îi corespunde, corespondenţa făcîndu-se prin numele variabilei-contor de buclă.

NOT

Format:

NOT *condiţie*

unde condiţie este o expresie care prin evaluare poate fi falsă sau adevărată (prin evaluarea operatorilor operaţionali sau logici).

NOT este funcţie şi returnează:

a)fals dacă *condiţie* este adevărată

b)adevarat dacă *condiţie* este falsă

OPEN#

Comandă microdrive

OR

Format:

condiție-1 **OR** *condiție-2*

unde *condiție-1/2* sînt expresii care prin evaluare pot fi false sau adevărate (prin evaluarea operatorilor operaționali sau logici).
OR este funcție logică și returnează fals dacă *condiție-1/2* sînt amîndouă false, altfel adevărat.

OUT

Format:

OUT *expresie-numerică-pozitivă-1*, *expresie-numerică-pozitivă-2*

OUT este comandă și permite trimiterea unui byte dat de evaluarea *expresiei-numerice-2* la portul de adresă *expresie-numerică-1*; expresiile fiind eventual rotunjite la numărul întreg cel mai apropiat.
Adrese de porturi:

254	BIN D7D6D5D4D3D2D1D0:	D4 difuzor D3 microfon D2D1D0 culoare BORDER
251	imprimantă	
247,239	porturi rezervate	

OVER

Format:

OVER *expresie-numerică-pozitivă*

OVER este comandă/operator și permite supraimprimarea conform valorii expresiei-numerice, eventual rotunjită la numărul întreg cel mai apropiat.

1 permite supraimprimarea
0 imprimare normală

PAPER

Format:

PAPER *expresie-numerică*

PAPER este comandă/operator și permite controlul culorii fondului ecranului conform expresiei numerice care eventual este rotunjită la valoarea întreagă cea mai apropiată.

Culori:	0 negru	5 cyan
	1 albastru	6 galben
	2 roșu	7 alb
	3 magenta	8 transparent
	4 verde	9 contrast

transparent: culoarea PAPER nu se modifică la afișări ulterioare
contrast: culoarea PAPER este negru/alb vizibil pe culoarea INK

PAUSE

Format:

PAUSE *expresie-numerică*

PAUSE este comandă și permite oprirea rulării programului pentru un timp determinat de numărul rezultat în urma evaluării expresiei numerice care eventual este rotunjită la numărul întreg cel mai apropiat.

număr: 0 pauză pînă la apăsarea unei taste (nu BREAK)
n timpul de așteptare este t=n*20ms (n între 1 și 65535)

PEEK

Format:

PEEK *constantă/variabilă-numerică* sau
PEEK (*expresie-numerică*)

PEEK este funcţie şi returnează ca valoare conţinutul numeric al byte-ului de adresă dată de *expresie-numerică* care eventual este rotunjită la numărul întreg cel mai apropiat.

PI

Format:

PI

PI este funcţie şi returnează ca valoare numărul PI (3.1415927).

PLOT

Format:

PLOT *listă-opţiuni expresie-numerică-1, expresie-numerică-2*

unde *listă-opţiuni* este opţională şi este de forma:

opţiune-afişare ; ...

opţiune-afişare este de tipul: PAPER, INK, BRIGHT, OVER,, FLASH sau INVERSE.

PLOT este comandă şi desenează pe ecran un punct de coordonate (X= *expresie-numerică-1*, Y= *expresie-numerică-2*), unde *expresiile-numerice* sînt eventual rotunjite la numărul întreg cel mai apropiat.

X măsurat de la stînga la dreapta, de la 0 la 255

Y măsurat de jos în sus, de la 0 la 175

POINT

Format:

POINT (*expresie-numerică-1, expresie-numerică-2*)

POINT este funcţie şi returnează: 0 pentru pixel de culoare PAPER (stins)
1 pentru pixel de culoare INK (aprins)

unde pixel-ul are coordonatele: X=*expresie-numerică-1*; Y=*expresie-numerică-2*

expresiile-numerice fiind eventual rotunjite la numărul întreg cel mai apropiat

X măsurat de la stînga la dreapta, de la 0 la 255

Y măsurat de jos în sus, de la 0 la 175

POKE

Format:

POKE *expresie-numerică-1, expresie-numerică-2*

POKE este comandă şi permite încărcarea unui byte de adresă dată de *expresie-numerică-1* cu valoare dată de *expresie-numerică-2*; eventual *expresiile numerice* sînt rotunjite la numărul întreg cel mai apropiat.

Observaţie: bytes avînd adresele între 0 şi 16383 (memoria ROM) nu pot fi modificaţi.

PRINT

Format:

PRINT *listă-opţiuni element-afişare separator* ...

listă-opţiuni este de forma:

opţiune-afişare;...

unde *opţiune-afişare* este de tipul: AT, TAB, PAPER, INK, BRIGHT, FLASH sau INVERSE; USR.

element-afişare este de forma:

expresie-numerică / şir

separator este:

; poziţia următoare de afişare este în continuare

, poziţia următoare de afişare este în coloana 0/16

' poziţia următoare de afişare este în linia următoare

PRINT este comandă şi permite afişarea pe ecran.

RANDOMIZE (RAND)

Format:

RANDOMIZE *expresie-numerică* (a)sau
RANDOMIZE USR *expresie-numerică* (b)

RANDOMIZE este comandă şi execută:

(a) introducerea *expresiei-numerice* opţionale, eventual rotunjită la numărul întreg cel mai apropiat, în variabila de sistem SEED (utilizată de RND); dacă *expresia-numerică* lipseşte sau este 0 în SEED se introduce variabila de sistem FRAMES (ceas de timp real) modulo 65536.

(b) salt la rutina în cod-maşină care începe la adresa dată de *expresia-numerică*

READ

Format:

READ *variabilă-numerică/şir* , ...

READ este comandă şi atribuie *variabilelor-numerice/ şir* valorile din instrucţiunile DATA începînd din locul unde s-a efectuat ultima citire READ anterioară (prima citire se face din prima instrucţiune DATA) dacă nu există o comandă RESTORE. Toate instrucţunile DATA sînt considerate ca formînd un singur şir de date (în ordinea numerelor de linie).

REM

Format:

REM *text*

REM este comandă şi permite introducerea comentariilor în programe. La rulare instrucţiunea REM are efect nul.

RESTORE

Format:

RESTORE *expresie-numerică-pozitivă*

RESTORE este comandă şi permite forţarea citirii READ de la începutul unei instrucţiuni DATA al cărei număr de linie este dat de *expresia-numerică*, eventual rotunjită la numărul întreg cel mai apropiat. Dacă nu există un astfel de număr atunci se citeşte din instrucţiunea DATA, avînd numărul de linie imediat următor. *Expresie-numerică* este opţională, valoare implicită/iniţială: 0.

RETURN

Format:

RETURN

RETURN este comandă şi permite reîntoarcerea în programul apelant după un apel GO SUB.

RND

Format:

RND

RND este funcţie şi returnează ca valoare un număr pseudo-aleator calculat cu formula:

75*(SEED+1)/65536-INT(75*(SEED+1)/65536)

unde SEED este variabilă-sistem.

RUN

Format:

RUN *expresie-numerică-pozitivă*

RUN este comandă şi permite lansarea în execuţie a programelor de la numărul de linie dat de *expresia-numerică*, eventual rotunjită la numărul întreg cel mai

apropiat. Dacă nu există un astfel de număr programul începe la numărul de linie imediat următor. Expresia-numerică este opţională, valoare implicită: 0.

SAVE

Format:

SAVE "*expresie-şir* " *opţiuni-date*

unde *opţiuni-date* sînt opţionale şi sînt de tipul CODE, DATA şi SCREEN$. SAVE este comandă şi permite salvarea fişierelor pe bandă magnetică. Dacă *opţiuni-date* lipsesc se salvează programul BASIC împreună cu valorile variabilelor sale, dacă acestea nu au fost şterse cu CLEAR.

SCREEN$

Format:

SCREEN$ (*linie, coloană*) (a)sau
SCREEN$ (b)

SCREEN$ este funcţie/operator care realizează:

(a) funcţie care returnează caracterul-şir, aflat în ecran în poziţia (*linie, coloană*). *linie* este numărul liniei, între 0 şi 21 de sus în jos *coloană* este numărul coloanei, între 0 şi 31 de la stînga la dreapta *linie* şi *coloană* pot fi constante, variabile sau expresii numerice eventual rotunjite la numărul întreg cel mai apropiat.

(b) opţiune utilizată în comenzile LOAD, SAVE sau VERIFY pentru manipularea fişierului imagine-ecran pe bandă magnetică.

SGN

Format:

SGN *constantă/variabilă-numerică* sau
SGN (*expresie-numerică*)

SGN este funcţie şi returnează semnul numărului:

1 dacă numărul este pozitiv
0 dacă numărul este 0
-1 dacă numărul este negativ

SIN

Format:

SIN *constantă/variabilă-numerică*
SIN (*expresie-numerică*)

SIN este funcţie şi returnează valoarea sinusului numărului considerat unghi măsurat în radiani.

SQR

Format:

SQR *constantă-variabilă-numerică-pozitivă*
SQR (*expresie-numerică-pozitivă*)

SQR este funcţie şi returnează ca valoare rădăcina pătrată (radicalul) numărului (pozitiv).

STEP

Format:

STEP *expresie-numerică*

STEP este operator şi permite stabilirea pasului de incrementare a variabilei de ciclare FOR, acesta fiind numărul dat de *expresia-numerică.*

STOP

Format:

STOP

STOP este comandă şi realizează oprirea rulării programului în locul respectiv.

STR$

Format:

STR$ *constantă/variabilă-numerică* sau
STR$ (*expresie-numerică*)

STR$ este funcţie şi permite transformarea numerelor în şiruri.

TAB

Format:

TAB *coloană*

TAB este operator şi stabileşte poziţia de afişare în comenzile INPUT şi PRINT. *coloană* este numărul coloanei, între 0 şi 31, numerotarea făcîndu-se de la stînga la dreapta. *coloana* poate fi constantă, variabilă sau expresie numerică pentru care se efectuează operaţiile de rotunjire la numărul întreg cel mai apropiat apoi modulo 32 dacă este cazul.

TAN

Format:

TAN constantă/variabilă-numerică sau
TAN (*expresie-numerică*)

TAN este funcţie şi returnează valoarea tangentei numărului considerat unghi măsurat în radiani.

THEN

Format:

THEN *instrucţiune* : ...

THEN este operator utilizat de IF care realizează secvenţa de comenzi care urmează, dacă condiţie este adevarată.

TO

Format:

TO *expresie-numerică* (a)
constantă/variabilă-şir (*expresie-numerică-1* **TO** *expresie-numerică-2*)(b)
(*expresie-şir*)(*expresie-numerică-1* **TO** *expresie-numerică-2*) (c)

TO este operator/funcţie şi realizează:

(a) operator în FOR, *expresie-numerică* reprezintă valoarea limita a variabilei de ciclare

(b),(c) funcţie de separare a subşirurilor. Din şirul dat de expresie se separă subşirul care începe cu caracterul cu numărul de ordine dat de *expresie-numerică-1* şi se termină cu caracterul cu numărul de ordine dat de *expresie-numerică-2* eventual expresiile numerice fiind rotunjite la numerele întregi cele mai apropiate. *expresie-numerică-1/2* sînt opţionale, valori implicite: primul/ultimul caracter al şirului.

USR

Format:

USR *constantă/variabilă-numerică* (a)sau
USR (*expresie-numerică*) (b)sau
USR *constantă/variabilă-şir* (c)

USR este operator/funcţie şi realizează:

(a),(b) apelul unei subrutine în cod-maşină care începe la adresa specificată de număr. *Expresie-numerică* este eventual rotunjită la numărul întreg cel mai apropiat. Dacă se doreşte reîntoarcerea în programul BASIC subrutina trebuie să se termine cu instrucţiunea RET, să nu modifice stiva şi IY iar HL' trebuie încărcat cu 2758h. Apelul poate fi făcut cu RANDOMIZE sau PRINT, (caz în care la retur se afişează conţinutul BC).

(c) funcţie care returnează adresa caracterului UDG respectiv

VAL

Format:

VAL *constantă/variabilă-şir*

VAL este funcţie şi returnează ca valoare expresie-numerică obţinută prin eliminarea ghilimelelor şirului.

VAL$

Format:

VAL$ *variabilă-şir*

VAL$ este funcţie şi returnează ca valoare expresie-şir obţinută prin eliminarea ghilimelelor din variabila şir.

VERIFY

Format:

VERIFY "*expresie-şir* " *opţiuni-date*

VERIFY este comandă şi permite verificarea fişierelor de pe bandă magnetică (salvate cu comanda LOAD) prin comparare cu cele aflate în memorie. *Expresie-şir* este opţională şi dacă lipseşte se verifică programe BASIC.

3.1.3 Structura programului BASIC

Un program BASIC are formatul:

linie-program-1
linie-program-2
...

O linie program are formatul:

număr-linie instrucţiune-1 : *instrucţiune-2* : ...

unde *număr-linie* poate fi un întreg între 0 şi 9999 şi instrucţiune are formatul:

comandă parametri-comandă

putînd fi de la 1 la 255 instrucţiuni într-o linie-program.

În mod normal la rulare liniile-program se execută în ordinea crescătoare a *numerelor-linie* şi în cadrul unei linii-program instrucţiunile se execută în ordine de la stînga la dreapta, cu excepţia instrucţiunilor GO SUB, GO TO, IF, LOAD, NEXT, RETURN, STOP.

Liniile-program introduse fără număr-linie sînt executate imediat.

3.1.4 Mesaje de eroare

Un mesaj de eroare are formatul:

cod-eroare mesaj, număr-linie : *număr-instrucţiune*

unde *cod-eroare* este codul erorii întîlnite 0 ... 9, A...R; *număr-linie* este numărul liniei în care a apărut eroarea; *număr-instrucţiune* este numărul instrucţiunii din linie în care a apărut eroarea; *mesaj* explică tipul erorii întîlnite.

0 OK

Program terminat normal
1 NEXT without FOR
Variabila de control NEXT nu a fost definită de FOR
2 Variabile not found
Variabilă nedefinită (lipseşte LET/INPUT/DIM)
3 Subscript wrong
Indice eronat la variabila-tablou (indexată)
4 Out of memory
Depăşire de memorie
5 Out of screen
Linie incorectă (23) în INPUT sau parametri incorecţi în PRINT AT
6 Number too big
Încercarea de a calcula un număr mai mare decît 10^38 generează eroare
7 RETURN without GO SUB
A fost întîlnită RETURN dar nu GO SUB anterior
8 End of file
Eroare microdrive
9 STOP statement
La execuţia unei instrucţiuni STOP
A Invalid argument
Argumentul unei funcţii matematice este incorect
B Integer out of range
O expresie numerică ce trebuie evaluată la intervalul (0, 255) sau (0, 65535) a generat o valoare în afara intervalului
C Nonsese in BASIC
Instrucţiune nevalidă
D BREAK-CONT repeats
A fost apăsată tasta BREAK, CONTINUE repetă instrucţiunea oprită
E Out of DATA
Instrucţiuni DATA insuficiente pentru READ sau lipseşte RESTORE
F Invalid file name
Nume fişier incorect pentru comanda SAVE
G No room for line
Nu există memorie suficientă pentru a introduce linia BASIC editată
H STOP in INPUT
În decursul efectuării instrucţiunii INPUT a fost introdusă comanda STOP, CONTINUE repetă INPUT.
I FOR without NEXT
Lipseşte instrucţiunea NEXT corespunzătoare
J Invalid I/O device
Eroare microdrive
K Invalid colour
Parametru incorect în PAPER, INK, BRIGHT, OVER, FLASH sau INVERSE
L BREAK into PROGRAM
A fost apăsată tasta BREAK, CONTINUE reia rularea programului de la instrucţiunea următoare

M RAMTOP no good
Valoarea specificată pentru variabila-sistem RAMTOP este incorectă (prea mare/mică)
N Statement lost
Tentativă de salt la o linie-program inexistentă
O Invalid stream
Eroare microdrive
P FN without DEF
Funcţia-utilizator respectivă nu a fost definită
Q Parameter error
Parametrii funcţiei-utilizator nu corespund; conform listei-variabile din definiţia ei
R Tape loading error
Eroare la citirea unui fişier de pe bandă magnetică, în LOAD/VERIFY

3.2 COMPILATORUL HIBASIC*

3.2.1 Instalarea

HIBASIC este cel mai bun compilator BASIC, fiind totodată şi complet. Instalarea se face prin simpla încărcare a programului.
Comenzi

*C	compilează programul BASIC
*R	rulează codul compilat
*X	şterge codul compilat
*T	listează, după rularea programului cu comanda RUN, numele variabilelor şi tipul lor
*ERASE	şterge programul BASIC (înlocuieşte NEW)
*D	analog *C, memorează numai DATA
*E	analog *C, nu memorează DATA

3.2.2 Directive

Directivele de compilare sînt puse în program la începutul acestuia şi au formatul:

număr-linie **REM :** *directivă*

unde directiva poate fi:

GO SUB *număr*

*HIBASIC este produs înregistrat al firmei HISOFT (Cameron Hayne)

declară subrutina care începe la linia *număr*
INT *variabilă-numerică*
declară variabila de tip număr întreg
INT+ *variabilă-numerică*
declară variabila de tip număr întreg pozitiv
LEN *variabilă-şir* < =*număr*
declară lungimea maximă a variabilei ca fiind *număr*, valoare implicită 255
LINE *număr*
afişează la compilare adresa liniei compilate
LIST
afişează la compilare variabilele
OPEN# *număr-linie-1*, **CLOSE#** *număr-linie-2*
defineşte prima respectiv ultima linie compilată

3.3 FIFTH*

3.3.1 Instalarea

Fifth este o extensie BASIC puternică specializată pe efecte grafice şi sonore deosebite.
Încărcarea programului se face cu secvenţa:
CLEAR 61029 : LOAD "" CODE
Fifth nu ocupă zona de memorie UDG.
Orice program care utilizeaza Fifth începe cu secvenţa:
RANDOMIZE *număr* **: RANDOMIZE USR 61030**
unde număr este numărul de bytes necesari comenzilor OBJECT.
Tasta Break poate fi dezactivată/activată cu POKE 65230,1/0.
Toate comenzile Fifth au formatul:
număr-linie **REM** *comandă parametri* \ ...
unde \ (back-slash) este separator de comenzi (analogul : din BASIC)
Comenzile sînt introduse literă cu literă şi separate de spaţii, nu se face distincţie între majuscule şi minuscule.

3.3.2 Erori

8 programul s-a terminat corect
0 salt la un număr de linie inexistent
Q comandă Fifth incorectă sau, normal, eroare BASIC

*Fifth este produs înregistrat al firmei Computer Rentals Ltd. (Richard M. Taylor)

A,B parametrii comenzii Fifth incorecţi sau, normal, eroare BASIC

3.3.3 Comenzi

*

Format:

REM * *text*

* permite introducerea comentariilor-text în instrucţiunea REM.

ALL

Format:

REM ALL *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

ALL stabileşte ca toate obiectele *nume-curent* (pentru toţi indicii) să devină curente (se lucrează în continuare cu ele).

COLOUR

Format:

PRINT *opţiune-afişare* ; ...

REM COLOUR *c/v-obiect* sau

REM TEMPS\COLOUR *c/v-obiect*

unde *opţiune-afişare* poate fi PAPER , INK , BRIGHT , FLASH , INVERSE sau OVER.

c/v-obiect este un şir (fără "") sau o variabilă-şir

COLOUR stabileşte culoarea de afişare pentru nume-obiect conform instrucţiunii PRINT vidă sau TEMPS; culoare 9 nu este utilizată.

DISABLE

Format:

REM DISABLE *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir.

DISABLE opreşte mişcarea obiectului curent pe ecran.

ENABLE

Format:

REM ENABLE *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir.

ENABLE reactivează mişcarea obiectului curent care a fost oprită cu DISABLE.

ERASE

Format:

REM ERASE *c/v-obiect*

unde *c/v-obiect* este variabilă-literă sau nume-obiect

În general un caracter este deplasat (MOVE) prin ştergerea lui din poziţia actuală (afişarea unui blank) apoi reafişarea caraterului în poziţia deplasată. Dacă caracterul are o margine de pixeli de lăţime egală sau mai mare cu numărul de pixeli de deplasare definiţi în SPEED (parametrul 3) atunci obiectul nu mai este şters înainte de deplasare. Fifth detectează automat această situaţie, mărind viteza de rulare a programului şi eliminînd efectul de clipire al caracterelor la afişare (dacă numărul lor este prea mare). Dar prin acest mod de lucru nu se detectează coliziunea obiectului respectiv cu altul. ERASE determină ştergerea caracterului din poziţia actuală înainte de afişarea lui în noua poziţie.

FILL

Format:

PRINT PAPER *expresie-numerică* ; **INK** *expresie-numerică*
REM FILL sau
REM TEMPS\FILL

FILL preia culorile definite în instrucţiunea PRINT vidă sau conform TEMPS şi le modifică pe tot ecranul fără a şterge conţinutul lui. Operatorii PAPER şi INK din PRINT sînt opţionali.

FIND

Format:

REM FIND *v/ff-1* , *v/ff-2*

unde *v/ff-1/2* sînt variabile-literă sau funcţii Fifth.
FIND determină numele obiectului aflat la poziţia dată.
v/ff-1 reprezintă coordonata x, de la 0 la 255, de la stînga la dreapta
v/ff-2 reprezintă coordonata y, de la 0 la 175, de jos în sus
Numele obiectului este returnat în variabila rezervată **j$**, dacă nu există nici un obiect se returnează şirul nul "". Sirul **j$** este format din litere minuscule.

GET

Format:

REM GET *v/ff-1* , *v/ff-2* , *v/ff-3* , *v/ff-4* , *variabilă-şir*

unde *v/ff* sînt variabile-literă sau funcţii Fifth.
GET salvează în *variabila-şir* setul de caractere afişate pe ecran într-un dreptunghi definit prin coordonatele sale:

v/ff-1 coordonata x a colţului stînga-sus
v/ff-2 coordonata y a colţului stînga-sus
v/ff-3 coordonata x a colţului dreapta-jos
v/ff-4 coordonata y a colţului dreapta-jos

unde coordonatele x reprezintă liniile, între 0 şi 21, de la stînga la dreapta; coordonatele y reprezintă coloanele, de la 0 la 31, de sus în jos; se salvează şi atributele de culoare.

INTERACT

Format:

REM INTERACT *v/ff*

unde *v/ff* este variabilă-literă sau funcţie Fifth.
INTERACT determină salt la o subrutină a programului în momentul în care 2 obiecte intră în coliziune (pe ecran ocupă acelaşi loc). Subrutina trebuie să se termine cu CONTINUE (nu RETURN), *v/ff* între 0 şi 65535, dacă *v/ff* >9999 acţiunea subrutinei este nulă. Sînt permise pînă la 16 interacţiuni simultane, altfel eroare.

INTPARAM

Format:

REM INTPARAM

INTPARAM returnează, pentru 2 obiecte intrate în coliziune (pe ecran ocupă acelaşi loc) în variabilele rezervate:
h$, i$ numele-obiectelor care se ciocnesc
h,i indexul respectiv al obiectelor
Dacă nu există ciocnire atunci şirurile sînt vide iar indexul nul

LARGE

Format:

PRINT *opţiune-afişare* ; ...

REM LARGE sau
REM TEMPS\LARGE

LARGE permite afişarea şirurilor de caractere mărite, la scară. Opţiunile selectate care pot fi PAPER, INK, BRIGHT, FLASH, OVER sînt stabilite în instrucţiunea PRINT vidă. TEMPS sau PRINT selectează culorile.
Variabilele rezervate utilizate de LARGE sînt:

a$ variabilă care conţine şirul de afişat
x variabilă care conţine poziţia x al colţului stînga-sus, măsurată de la stînga la dreapta,de la 0 la 255
y variabilă care conţine poziţia y al colţului stînga-sus, măsurată de sus în jos, de la 0 la 175
t variabilă care conţine scara pe verticală, între 1 şi 22
w variabilă care conţine scara pe orizontală, între 1 şi 32

Valorile numerice sînt rotunjite la numărul întreg cel mai apropiat.

LET

Format:

REM LET *variabilă-literă =funcţie-Fifth*

LET atribuie unei variabile-literă numerice valoarea funcţiei Fifth (în scopul prelucrării ulterioare în expresii).

LIMIT

Format:

REM LIMIT *v/ff*

unde *v/ff* este variabilă-literă sau funcţie Fifth.
LIMIT determină salt la o subrutină a programului în momentul în care obiectul a atins marginea ecranului. Subrutina trebuie să se termine cu CONTINUE (nu RETURN). *v/ff* între 0 şi 65535, dacă *v/ff* >9999 acţiunea subrutinei este nulă. Sînt permise pînă la 16 obiecte care ating simultan limita, altfel eroare.

LTMPARAM

Format:

REM LTMPARAM

LTMPARAM returnează, pentru un obiect care a atins limita ecranului:

în **i** 0=margine-sus
1=margine-dreapta
2=margine-jos
3=margine-stînga
în **h$** *nume-obiect*
în **h** indexul *numelui-obiect*

i,h$,h sînt variabile rezervate.

MOVE

Format:

REM MOVE *c/v-obiect , v/ff-1 , v/ff-2*

unde: *c/v-obiect* este un şir (fără "") sau o variabilă-şir
v/ff-1/2 sînt variabile literă sau funcţii Fifth

MOVE deplasează obiectul curent conform VECTOR şi SPEED, doar în timpul rulării programului, pînă la marginea ecranului, de la poziţiile iniţiale *v/ff-1/2*. MOVE poate fi folosită şi pentru a deplasa obiecte într-o buclă FOR-NEXT deoarece şterge imaginea anterioară.
v/ff-1 reprezintă coordonata x, de la 0 la 255, de la stînga la dreapta
v/ff-2 reprezintă coordonata y, de la 0 la 175, de jos în sus

OBJECT

Format:

REM OBJECT *c/v-obiect* , *v/ff*

unde: *v/ff* este variabila-litera/functie-Fifth

c/v-obiect este un şir (fără "") sau o variabilă-şir

OBJECT defineşte un set de obiecte, de tip tablou unidimensional, al căror număr este dat de expresia-Fifth, care pot fi manipulate pe ecran (analog DIM). Obiectele sînt caractere alfanumerice, grafice sau UDG, cu excepţia blank-ului; toate obiectele dintr-un tablou fiind de acelaşi tip. Numele reprezintă un şir alfanumeric.

Spaţiul alocat obiectelor este definit de prima instrucţiune a programului; memoria ocupată de un obiect se calculează cu expresia:

lungime-nume+**10**+**6***dimensiune-tablou*

PRINT

Format:

REM PRINT *c/v-obiect* , *caracter*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

PRINT defineşte obiectul ca fiind caracter. Sînt permise toate caracterele alfanumerice, grafice şi UDG cu excepţia blank-ului.

PUT

Format:

REM PUT *v/ff-1*, *v/ff-2*, *variabilă-şir*

unde *v/ff* sînt variabile-literă sau funcţii Fifth.

PUT afişează pe ecran variabila-şir salvată cu GET în poziţia:

v/ff-1 coordonata x a colţului stînga-sus a dreptunghiului; linia între 0 şi 21, de la stînga la dreapta

v/ff-2 coordonata y a colţului stînga-sus a dreptunghiului; coloana, între 0 şi 31, de sus în jos

variabilă-şir şirul de afişat, salvat anterior cu GET

PUT poate fi precedată de TEMPS sau o instrucţiune PRINT vidă care stabileşte culorile de afişare.

Culoare: 8 PUT reface culorile conform salvării GET
9 asigură contrastul PAPER/INK
INVERSE 1 inversează culorile din salvarea GET

REPLACE

Format:

PRINT PAPER *expresie-numerică* ; **INK** *expresie-numerică*
REM REPLACE sau
REM REPLACE\FILL

REPLACE înlocuieşte culorile temporare specificate în instrucţiunea PRINT vidă cu culorile permanente, selectiv, în toate locurile de pe ecran unde acestea există (dacă există). TEMPS\REPLACE înlocuieşte culorile permanente tot cu ele.

Culoare: 8 culoarea respectivă PAPER/INK este ignorată la căutare
9 fără utilizare

RMOVE

Format:

REM RMOVE *c/v-obiect* , *v/ff-1* , *v/ff-2*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

v/ff-1/2 sînt variabile-literă sau funcţii Fifth

RMOVE deplasează obiectul curent conform deplasărilor relative *v/ff-1/2* faţă de o poziţie anterioară.

v/ff-1 reprezintă coordonata x, de la 0 la 255, de la stînga la dreapta
v/ff-2 reprezintă coordonata y, de la 0 la 175, de jos în sus
Obiectul nu poate ieşi din ecran: o coordonată relativă mai mare decît 255/175 reprezintă o deplasare relativă în sens contrar (coordonate circulare). RMOVE sterge imaginea anterioară (poate fi folosită într-o buclă FOR-NEXT pentru deplasarea obiectelor).

SOUND

Format:

REM SOUND *v/ff-1* , *v/ff-2* , *v/ff-3* , *v/ff-4*

unde *v/ff* sînt variabile-literă sau funcţii Fifth.
SOUND emite o serie de sunete, cu tonalitaţi (în creştere sau în descreştere).
v/ff-1 reprezintă numărul de sunete din serie, între 0 şi 255
v/ff-2 reprezintă durata, între 0 şi 65535
v/ff-3 reprezintă tonul iniţial, între 0 şi 65535
v/ff-4 reprezintă treapta de incrementare, între 0 şi 32767 pozitivă, între 65535 şi 32768 negativă.

SPEED

Format:

REM SPEED *c/v-obiect* , *v/ff-1* , *v/ff-2*

unde *v/ff-1/2* sînt variabile-literă sau funcţii Fifth
c/v-obiect este un şir (fără "") sau o variabilă-şir
SPEED stabileşte viteza de deplasare a obiectului curent:
v/ff-1 reprezintă timpul după care obiectul este deplasat: *v/ff* **-1*20**ms; între 0 şi 255
v/ff-2 reprezintă numărul de pixeli săriţi de obiect la fiecare deplasament; între 0 şi 255

TEMPS

Format:

TEMPS

TEMPS stabileşte culorile (temporare) de afişare pentru următoarele comenzi Fifth, conform celor stabilite anterior prin comenzile PAPER, INK.

USE

Format:

REM USE *c/v-obiect* , *v/ff*

unde *c/v-obiect* este un şir (fără "") sau funcţie Fifth
v/ff este o variabilă-literă sau funcţie Fifth
USE stabileşte obiectul curent şi indicele curent (cu care se va lucra în continuare).

VECTOR

Format:

REM VECTOR *c/v-obiect* , *v/ff*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir
v/ff este variabilă-literă sau funcţie Fifth
VECTOR stabileşte direcţia următoare de deplasare pentru obiectul curent (definit de USE sau ALL).
Există 16 direcţii de deplasare, numerotate de la 0 la 15: 0 este în sus, următoarea direcţie este decalată faţă de cea anterioară cu 22.50 (2PI/16 rad) în sens orar.

3.3.4 Funcţii

Funcţiile Fifth sînt utilizate ca parametri în comenzile Fifth.

ATTR

Format:

ATTR *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

ATTR returnează ca valoare codul de atribute-culoare al obiectului în formatul ATTR BASIC Spectrum.

COLUMN

Format:

COLUMN *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

COLUMN returnează ca valoare numărul coloanei în care se găseşte obiectul curent, definit de USE.

CURRENT

Format:

CURRENT *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

CURRENT returnează indicele curent al obiectului. Sub ALL (toţi indicii sînt curenţi) se returnează valoarea 0.

DIRECTION

Format:

DIRECTION *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

DIRECTION returnează ca valoare direcţia de deplasare a obiectului (parametrul 2, VECTOR).

INTERACT

Format:

INTERACT

INTERACT returnează numărul liniei unde începe subrutina de tratare a comenzii INTERACT. Dacă *număr*> =9999 atunci subrutina nu există.

LIMIT

Format:

LIMIT

LIMIT returnează numărul liniei unde începe subrutina de tratare a comenzii LIMIT. Dacă *număr*> =9999 atunci subrutina nu există.

LINE

Format:

LINE *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

LINE returnează ca valoare numărul liniei în care se găseşte obiectul curent, definit de USE.

MASK

Format:

MASK *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir

MASK returnează ca valoare byte-ul de mascare pentru culoare 8 (bit 1 înseamnă culoare nemodificată).

NO

Format:

NO *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir
NO returnează ca valoare numărul obiectelor-nume (parametrul 2, OBJECT).

SCREEN

Format:

SCREEN *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir
SCREEN returnează ca valoare codul caracterului-obiect.

STATUS

Format:

STATUS *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir
STATUS returnează ca valoare, pentru obiectul curent (USE):

1 dacă ENABLE
0 dacă DISABLE

VELOCITY

Format:

VELOCITY *c/v-obiect*

unde *c/v-obiect* este un şir (fără "") sau o variabilă-şir
VELOCITY returnează ca valoare numărul de pixeli săriţi de obiectul curent (USE) la fiecare deplasament (MOVE) (parametrul 3, SPEED).

3.4 BETA BASIC*

3.4.1 Instalarea

Beta BASIC este o extensie BASIC puternică care introduce o serie de comenzi şi functii noi, accesibile analog cuvintelor-cheie BASIC Spectrum, precum şi facilităţi de structurare şi editare a programelor. Instalarea se face simplu, cu **LOAD** "". Pentru dezactivarea sunetului de tastă apăsată se utilizeaza POKE 23609,0. Beta BASIC introduce o linie 0, rezidentă în toate programele,de definiţie a comenzilor şi funcţiilor proprii astfel încît toate programele care rulează sub Beta BASIC trebuie încarcate cu **MERGE** "".

*Beta Basic este produs înregistrat al firmei Beta Soft

3.4.2 Comenzi

Cuvintele-cheie comenzi sînt obţinuţe, în general, în modul cursor G (GRAPHICS) prin apăsarea unei taste literare sau cifre.

ALTER (G:A)

Format:

ALTER *listă-opţiuni-1* **TO** *listă-opţiuni-2*

unde *lista-optiuni* are formatul:

opţiune-afişare , ...

şi *opţiune-afişare* este de tipul PAPER, INK, BRIGHT sau FLASH. ALTER modifică atributele de culoare pe tot ecranul, fără să îl şteargă:

a) conform *listă-opţiuni-2* dacă *listă-opţiuni-1* lipseşte
b) înlocuieşte atributele de culoare din *listă-opţiuni-1* cu cele din *listă-opţiuni-2*

AUTO

Format:

AUTO *număr-întreg-1* , *număr-întreg-2*

AUTO introduce facilitatea de editare automată a numerelor de linie începînd de la *număr-întreg-1* cu pasul *număr-întreg-2*.
Numerele-întregi-1/2 sînt opţionale, valori implicite 10, 10.
Dacă în AUTO se modifică numărul de linie, acesta este preluat în continuare de AUTO. Ieşirea din AUTO se face cu tasta BREAK apăsată minim 1s.

BREAK-*(SS şi blank)

Apăsarea tastelor SS şi blank (spaţiu) minim 1s determină oprirea orcărui program care rulează sub Beta BASIC şi afişarea cursorului *. Pentru aceasta trebuie ca programul să nu dezactiveze întreruperile mascabile.

CLOCK (G:C)

Format:

CLOCK *expresie-numerică* (a)sau
CLOCK *expresie-şir* (b)

CLOCK efectuează servicii de ceas real: afişeaza timpul curent, alarma sonoră la un moment determinat, apelul unei subrutine la un moment determinat.

(a) dacă numărul este între 0 şi 7 acesta defineşte modul de lucru:

număr	subrutină	alarmă	afişare
0	nu	nu	nu
1	nu	nu	da
2	nu	da	nu
3	nu	da	da
4	da	nu	nu
5	da	nu	da
6	da	da	nu
7	da	da	da

dacă numărul este între 8 şi 9999 acesta reprezintă numărul de linie unde începe subrutina de apelat în momentul în care timpul curent coincide cu timpul-alarmă

(b) poziţionează ceasul:
cifra-1 cifra-2 (ora): *cifra-3 cifra-4* (minutul): *cifra-5 cifra-6* (secunda)
conform primelor 6 cifre din şir, alte caractere fiind ignorate, cu excepţia "a" , "A" ; cifrele lipsă se consideră 0 ; "a" , "A " la începutul şirului

semnalizează faptul că timpul care urmează reprezintă timpul alarmă.

CURSOR (CHR$ 8, 9, 10, 11)

Format:

CHR$ *expresie-numerică*

Bug-urile din codurile-cursor:

8 cursor-stînga 9 cursor-jos
10 cursor-jos 11 cursor-sus

au fost corectate.

DEF KEY (G:SS şi 1)

Format:

DEF KEY "*literă/cifră* " ; *expresie-şir* (a)sau
DEF KEY "*literă/cifră*" ; *instrucţiune-1* : ... (b)

DEF KEY introduce facilităţi de editare. Prin apăsarea tastei literă sau cifră definite, în modul cursor * (SS şi blank), se introduce:

(a) şirul dat de *expresie-şir*
(b) şirul de instrucţiuni definite.

Prin redefinirea unei taste vechea definiţie este ştearsă. NEW nu şterge definiţiile, DEF KEY ERASE şterge toate definiţiile.

DEF PROC (G:1)

Format:

DEF PROC *expresie-şir*

DEF PROC marchează începutul definiţiei unei proceduri al cărui nume este *şir*. În numele şirului blank-urile sînt ignorate şi nu se face distincţia între literele majuscule şi minuscule. Se face distincţia între numele unei proceduri şi numele unei variabile cu acelaşi nume. DEF PROC trebuie să fie prima instrucţiune din linia BASIC. DEF PROC poate fi plasată oriunde în program şi dacă se termină cu un unic END PROC nu este executată decît la apelul PROC corespunzător. Variabilele din DEF PROC sînt comune întregului program.

DELETE (G:7)

Format:

DELETE *număr-1* **TO** *număr-2*

DELETE şterge liniile BASIC avînd numerele între *număr-1* şi *număr-2* care sînt opţionale. Dacă *număr-1* lipseşte ştergerea se efectuează pînă la ultima linie. *număr-1* şi *număr-2* trebuie să fie numerele unor linii existente în program. Variabilele BASIC nu sînt şterse.

DO(G:D)

Format:

DO (a)sau
DO WHILE *condiţie* (b)sau
DO UNTIL *condiţie* (c)

DO marchează începutul intrării într-o buclă fără contor care se execută:

(a) necondiţionat
(b) atîta timp cît *condiţie* este adevărată
(c) atîta timp cît *condiţie* este falsă

Sfîrşitul buclei este marcat de LOOP. Sînt permise buclele imbricate, sînt interzise buclele încrucişate. Ieşirea din buclă la mijlocul ei se poate face numai cu EXIT IF sau POP.

DPOKE (G:P)

Format:

DPOKE *expresie-numerică-1* , *expresie-numerică-2*

DPOKE plasează în memorie la adresa dată de *expresie-numerică-1* byte-ul inferior al *expresiei-numerice-2* şi la adresa imediat următoare (+1) byte-ul superior.

EDIT (Enter apoi 0)

Format:

EDIT *expresie-numerică*

EDIT lansează direct în editare linia a cărei număr este dat de *expresia- numerică* indiferent de poziţia cursorului în listing. *expresie-numerică* este opţională, valoarea implicită este numărul de linie indicat de cursor în listing.

ELSE(G:E)

Format:

ELSE *instrucţiune-1* : ...

ELSE defineşte instrucţiunile care se execută în IF dacă condiţia este falsă, ELSE trebuie plasată în aceeaşi linie cu IF, THEN şi este precedată de : . ELSE este opţională şi în cazul condiţionalelor IF imbricate aparţine celei mai apropiate IF anterioare.

END PROC (G:3)

Format:

END PROC

END PROC marchează sfîrşitul definiţiei unei proceduri (DEF PROC) şi reîntoarcerea la programul apelant. O definiţie de procedură poate avea mai multe puncte de ieşire (retur) dar în acest caz ea trebuie separată de restul programului deoarece nu se mai asigură neexecutarea ei.

EXIT IF (G:I)

Format:

EXIT IF *condiţie*

EXIT IF este utilizată în interiorul unei bucle DO-LOOP şi determină saltul la prima instrucţiune după LOOP (părăsirea buclei) dacă *condiţie* este adevărată.

FILL(G:F)

Format:

FILL *expresie-numerică-1* , *expresie-numerică-2* (a)sau
FILL INK *culoare* ; *expresie-numerică-1* ; *expresie-numerică-2* (b)sau
FILL PAPER *culoare* ; *expresie-numerică-1* ; *expresie-numerică-2* (c)

unde *culoare* este o expresie-numerică care reprezintă o culoare validă
şi *expresie-numerică-1/2* reprezintă coordonata punctului x/y de unde începe operaţia, punct oarecare în zonă:
x între 0 şi 255 de la stînga la dreapta
y între 0 şi 175 de jos în sus

FILL umple o zonă ecran delimitată de o curbă închisă sau marginile sale astfel:
(a) pixel culoare PAPER devine pixel culoare INK permanentă
(b) pixel culoare PAPER devine pixel culoare INK temporară
(c) pixel culoare INK devine pixel culoare PAPER

GET (G:G)

Format:

GET *variabilă-numerică/şir*

GET aşteaptă apăsarea unei taste şi atribuie variabilei:
numerice: valoarea cifrei apăsate (0 ... 9) sau un cod numeric pentru celelalte taste (A= 10 ... a=42...) conform modului cursor (L, C, E, G)
şir: şirul cu un unic element caracterul-tastă-apăsată conform modului cursor (L , C, E, G)

JOIN (G:SS şi 6)

Format:

JOIN *expresie-numerică*

JOIN concatenează la sfîrşitul liniei BASIC al cărei număr este dat de *expresie numerică* conţinutul liniei BASIC următoare

KEY IN (G:SS şi 4)

Format:

KEY IN *variabilă-şir*

KEY IN introduce o linie BASIC, în program, în timpul rulării acestuia, care este dată de *variabila-şir*. KEY IN nu poate fi utilizată decît în cadrul unui program.

KEYWORDS (G:8)

Format:

KEYWORDS *expresie-numerică*

KEYWORDS comută afişarea caractere grafice/cuvinte-cheie Beta BASIC conform numărului:

0 caractere grafice , cu excepţia KEYWORDS
1 cuvinte-cheie Beta BASIC

KEYWORDS 0 afectează listingul programului dar nu rularea (funcţionalitatea) acestuia.

LIST

Format:

LIST *expresie-numerică-1* **TO** *expresie-numerică-2*

LIST efectuează comanda BASIC Spectrum normală şi, conform formatului, listarea liniilor avînd numerele între *expresie-numerică-1* şi *expresie-numerică-2*. *Expresiile-numerice-1/ 2* sînt opţionale, valori implicite prima/ultima linie de program.

LLIST

Format:

LLIST *expresie-numerică-1* **TO** *expresie-numerică-2*

LLIST efectuează comanda BASIC Spectrum normală şi, conform formatului, listarea liniilor avînd numerele între *expresie-numerică-1* şi *expresie-numerică-2*. *Expresiile-numerice-1/ 2* sînt opţionale, valori implicite prima/ultima linie de program.

LOOP (G:L)

Format:

LOOP (a)sau
LOOP WHILE *condiţie* (b)sau
LOOP UNTIL *condiţie* (c)

LOOP marchează sfîrşitul unei bucle fără contor:

(a) utilizat de DO/DO LOOP/DO UNTIL
WHILE: dacă *condiţie* este adevărată se efectuează salt la prima instrucţiune de după DO, altfel continuă
UNTIL: dacă *condiţie* este falsă se efectuează salt la prima instrucţiune de după DO, altfel continuă

(b) utilizat de DO, dacă *condiţie* este adevărată se efectuează salt la prima instrucţiune de după DO , altfel continuă

(c) utilizat de DO, dacă *condiţie* este falsă se efectuează salt la prima instrucţiune de după DO, altfel continuă

Buclele DO-LOOP WHILE/UNTIL sînt parcurse cel puţin odată.

ON (G:O)

Format:

GO TO ON *expresie-numerică* ; *număr-linie-1* , ... sau
GO SUB ON *expresie-numerică* ; *număr-linie-1* , ...

unde *numerele-linie* sînt expresii numerice
ON determină, conform comenzii, salt la linia a cărei număr are ordinul dat de *expresia-numerică*. Dacă numărul nu se încadrează în intervalul **1**, ..., *numarul-de-elemente-în-listă*, comanda este ignorată.

ON ERROR(G:N)

Format:

ON ERROR *expresie-numerică*

ON ERROR permite mascarea erorilor prin apelarea unei subrutine (tip GO SUB), care începe la numărul de linie dat de *expresie-numerică*, la întîlnirea unei erori, cu excepţia erorilor 0 şi 9. Comanda este dezactivată prin **ON ERROR 0** şi în timpul rulării subrutinei proprii (de apel). ON ERROR utilizează 3 variabile-numerice rezervate:

LINE numărul liniei în care a intervenit eroarea
STAT numărul instrucţiunii din linie în care a intervenit eroarea
ERROR codul erorii intîlnite

PLOT

Format:

PLOT *listă-opţiuni* ; *expresie-numerică-1* ,
expresie-numerică-2 , *expresie-numerică-3*

PLOT efectuează comanda BASIC Spectrum normală şi, conform formatului, afişarea unui şir într-o poziţie dată. *Expresie-numerică-1/ 2* definesc coordonatele x/y ale colţului dreapta-sus al primului caracter al şirului:

x între 0 şi 255, de la stînga la dreapta
y între 0 şi 175, de jos în sus

Fragmentul de şir care depăşeşte marginea dreaptă a ecranului este afişat în stînga (coordonate circulare).

POKE

Format:

POKE *expresie-numerică* , *expresie-şir*

POKE efectuează comanda BASIC Spectrum normală şi, conform formatului, stocarea unui şir în memorie, începînd de la adresa dată (în formatul codurilor ASCII).

POP (G:C)

Format:

POP *variabilă-numerică*

POP retrage un număr de linie (adresa de retur) din stiva GO SUB/PROC/DO-LOOP. *Variabila-numerică* este opţională şi dacă este dată capătă valoarea numărului-de-linie. POP permite ieşirea din subrutine, proceduri, bucle fără încărcarea stivei respective.

PROC (G:2)

Format:

PROC *expresie-şir*

PROC apelează procedura definită cu DEF PROC respectiv.

RENUM (G:4)

Format:

RENUM (*număr-linie-1* **TO** *număr-linie-2*)
LINE *număr-linie-3* **STEP** *expresie-numerică*

RENUM renumerotează un bloc de linii-program care începe la *număr-linie-1* şi se termină la *număr-linie-2*. Prima linie a blocului primeşte *număr-linie-3*, următoarele primind numere de linie conform pasului dat de *expresie-numerică*. *Număr-linie-1/ 2* sînt opţionale, valori implicite 10, 10; dacă lipsesc expresia dintre paranteze poate lipsi.
LINE *număr-linie-3*, **STEP** *expresie-numerică* sînt opţionale, valori implicite: 10, 10.
RENUM acordează toate instrucţiunile care folosesc numere de linie, cu excepţia CLOCK, în conformitate cu noua numerotare, dacă numerele de linie sînt date explicit; variabilele/expresiile-numerice generează eroare Y şi vechea linie nu este ştearsă.

ROLL (G:R)

Format:

ROLL *expresie-numerică-cod* , *expresie-numerică-pixel* ;
expresie-numerică-x ; *expresie-numerică-y* ;
expresie-numerică-fereastră-x ; *expresie-numerică-fereastră-y*

ROLL efectuează deplasarea unei ferestre-ecran, ceea ce iese din ecran printr-o margine intră prin marginea opusă; conform codului:

cod	direcţie	deplasează
1	stînga	atribute
2	jos	atribute
3	sus	atribute
4	dreapta	atribute
5	stînga	pixeli
6	jos	pixeli
7	sus	pixeli
8	dreapta	pixeli
9	stînga	pixeli şi atribute
10	jos	pixeli şi atribute
11	sus	pixeli şi atribute
12	dreapta	pixeli şi atribute

Expresie-numerică-pixel este opţională, valoare implicită 1; defineşte numărul de pixeli cu care se deplasează fereastra.
Expresie-numerică-x, ..., *expresie-numerică-fereastră-y* sînt opţionale şi definesc fereastra de deplasat, implicit întregul ecran.

x,y coordonatele x , y ale colţului stînga-sus al ferestrei
x de la 0 la 255, de la stînga la dreapta
y de la 0 la 175, de jos în sus
fereastră-x lungimea x a ferestrei, în coloane
între 1 şi 32, de la stînga la dreapta
fereastră-y înalţimea y a ferestrei în pixeli
între 0 şi 176, de sus în jos

Atributele nu pot fi deplasate decît în paşi multiplu de 8 pixeli, indiferent de numărul de pixeli-deplasament.

SCROLL (G:S)

Format:

SCROLL *expresie-numerică-cod* , *expresie-numerică-pixel* ;
expresie-numerică-x ; *expresie-numerică-y* ;
expresie-numerică-fereastră-x ; *expresie-numerică-fereastră-y*

SCROLL efectuează deplasarea unei ferestre-ecran, ceea ce iese din ecran printr-o margine se pierde; conform codului:

cod	direcţie	deplasează
1	stînga	atribute
2	jos	atribute
3	sus	atribute
4	dreapta	atribute
5	stînga	pixeli
6	jos	pixeli
7	sus	pixeli
8	dreapta	pixeli
9	stînga	pixeli şi atribute
10	jos	pixeli şi atribute
11	sus	pixeli şi atribute
12	dreapta	pixeli şi atribute

Expresie-numerică-pixel este opţională, valoare implicită 1; defineşte numărul de pixeli cu care se deplasează fereastra.
Expresie-numerică-x, ..., expresie-numerică-fereastră-y sînt opţionale şi definesc fereastra de deplasat, implicit întregul ecran.

x, y coordonatele x, y ale colţului stînga-sus al ferestrei
x de la 0 la 255, de la stînga la dreapta
y de la 0 la 175, de jos în sus
fereastră-x lungimea x a ferestrei, în coloane
între 1 şi 32, de la stînga la dreapta
fereastră-y înaltimea y a ferestrei în pixeli
între 0 şi 176, de sus în jos

Atributele nu pot fi deplasate decît în paşi multiplu de 8 pixeli, indiferent de numărul de pixeli-deplasament.

SORT (G:M)

Format:

SORT *v-n/s opţiune-indice opţiune-indice* (a)sau
SORT INVERSE *v-n/s opţiune-indice opţiune-indice* (b)

unde *v-n/s* este o variabilă-numerică/şir
şi *opţiune-indice* sînt opţionale, de forma:
(*indice-1* **TO** *indice-2*) (1)sau
(*indice*) (2)sau
( ) (3)

SORT sortează variabile-tablou, uni sau bi-dimensionale, în ordine directă (a) sau inversă (b) (ordinea este: pentru numere: numerală; pentru şiruri: codul ASCII):
(1) sortează între *indice-1* şi *indice-2*; *indicii-1/2* sînt opţionali, valori implicite primul /ultimul element
(2) sortează partea de tablou definită de *indice*
(3) indicii (pe poziţia respectivă) sînt indiferenţi

SPLIT (SS şi W)

Introducerea caracterului < > într-o linie BASIC imediat după : determină separarea liniei în 2 parţi: prima este introdusă în listing iar a 2-a în zona editare, amîndouă cu numărul de linie respectiv.: trebuie sa nu fie precedat de o instrucţiune REM.

TRACE (G:T)

Format:

TRACE *expresie-numerică*

TRACE permite depanarea programelor, apelînd o subrutină înaintea execuţiei fiecarei instrucţiuni BASIC. Numărul defineşte:

0 dezactivează TRACE, rulare normală
1...9999 activează TRACE, numărul reprezintă numărul de linie de unde începe subrutina de apelat

TRACE folosește 2 variabile rezervate:
LINE numărul liniei curente
STAT numărul instrucțiunii curente din linia curentă

UNTIL (G:K)

Format:

UNTIL *condiție*

Utilizat în DO/LOOP determină execuția ciclului atîta timp cît condiție este falsă.

USING (G:U)

Format:

USING *format* ; *constantă/variabilă-numerică*

unde format este #...#.#... sau 0...0.0...

USING este utilizat în PRINT pentru a stabili formatul de afișare al numerelor: numărul de cifre înainte și după . .

suprimă 0-urile (redundante) din fața numărului
0 afișează 0-urile (redundante) din fața și coada numărului
% semnalizează depășire, cifrele inferioare (mai putin semnificative) sînt suprimate conform formatului, ultima cifră afișată este eventual rotunjită.

WHILE (G:J)

Format:

WHILE *condiție*

Utilizat în DO/LOOP determină execuția ciclului atîta timp cît condiție este falsă.

XOS, YOS, XRG, YRG

Variabile rezervate pentru comenzile PLOT, DRAW, CIRCLE, FILL:
XOS originea x a sistemului de coordonate, inițial 0 (stînga jos)
YOS originea y a sistemului de coordonate, inițial 0 (stînga jos)
XRG scala axei x, inițial 256
YRG scala axei y, inițial 176

3.4.3 Funcții

Funcțiile se obțin prin tastarea **FN** *literă simbol* unde *simbol* este (pentru funcții-numerice sau **$** pentru funcțiile-șir.
În momentul tastării simbolului **FN** *litera* se șterge și este înlocuită cu denumirea funcțiilor. Funcțiile sînt definite într-o linie 0 introdusă de Beta BASIC în toate programele.

AND (FN A()

Format:

AND (*expresie-numerică-1* , *expresie-numerică-2*)

AND efectuează operația logică și pe biți între *expresie-numerică-1* și *expresie-numerică-2* care pot fi între 0 și 65535.

BIN$ (FN B()

Format:

BIN$ (*expresie-numerică*)

Returnează un şir format din 0 şi 1, cu 8 sau 16 caractere, care reprezintă forma binară a *expresiei-numerice* evaluată la un byte (8 biţi) sau 2 bytes (16 biţi).

CHAR$ (FN C$)

Format:

CHAR$ (*expresie-numerică*)

CHAR$ transformă *expresia-numerică*, între 0 şi 65535, într-un şir de 2 caractere conform codurilor ASCII. Primul caracter are codul: **INT** (*expresie-numerică* / **256**), al 2-lea caracter are codul *expresie-numerica*—**256*INT**(*expresie-numerică* /**256**).

COSE (FN C()

Format:

COSE (*expresie-numerică*)

COSE returnează cosinusul unghiului exprimat în radiani cu o aproximaţie de numai 4 zecimale exacte dar aproximativ de 6 ori mai rapid decît COS.

DEC (FN D()

Format:

DEC (*expresie-şir*)

DEC converteşte într-un număr zecimal şirul citit ca un număr hexazecimal cu minim 1 şi maxim 4 cifre.

DPEEK (FN P()

Format:

DPEEK (*expresie-numerică*)

DPEEK citeşte un număr pe 2 bytes din memorie de la adresa dată de *expresie-numerică*, în convenţia Intel, astfel încît valoarea este **PEEK** (*expresie-numerică*) **+256 * PEEK** (*expresie-numerică* **+1**).

FILLED (FN F()

Format:

FILLED ()

FILLED returnează numărul de pixeli manipulaţi de ultima comandă FILL.

HEX (FN H$)

Format:

HEX$ (*expresie-numerică*)

HEX$ converteşte un număr zecimal, între -255 şi 65535, în echivalentul său hexazecimal (pentru numere negative în complement faţă de 2).

INSTRING (FN I()

Format:

INSTRING (*expresie-numerică* , *expresie-şir-1* , *expresie-şir-2*)

INSTRING caută în şirul dat de *expresie-şir-1* subşirul dat de *expresie-şir-2*, începînd cu caracterul al cărui număr de ordine este dat de *expresie-numerică* şi returnează poziţia primului caracter al subşirului sau 0 dacă acesta nu a fost găsit. # poate înlocui în subşir orice caracter, cu excepţia primului.

MEM (FN M()

Format:

MEM ()

MEM returnează numărul de bytes liberi din memorie, realizînd aceeaşi funcţie ca **PRINT 65535—USR 7962** (care returnează cu 8 bytes mai mult).

MEMORY$ (FN M$)

Format:

MEMORY$()

MEMORY$ returnează conținutul memoriei, mai puțin prima și ultimele 3 locații, ca un șir, astfel încît are lungimea 65532.

MOD (FN V()

Format:

MOD (*expresie-numerică-1* , *expresie-numerică-2*)

MOD returnează restul împărțirii (întregi) *expresie-numerică-1* la *expresie-numerică-2*.

NUMBER (FN N()

Format:

NUMBER (*expresie-șir*)

NUMBER transformă un șir de 2 caractere într-un număr, între 0 și 65535, conform codurilor ASCII:

număr =**256*CODE** *șir (1)* +**CODE** *șir (2)*

OR (FN O()

Format:

OR (*expresie-numerică-1* , *expresie-numerică-2*)

OR efectuează operația logică sau între *expresie-numerică-1* și *expresie-numerică-2* care pot fi între 0 și 65535.

RNDM (FN R()

Format:

RNDM (*expresie-numerică*)

RNDM furnizează un număr aleator întreg între 0 și *expresie-numerică* eventual rotunjită la numărul întreg cel mai apropiat. Dacă *expresie-numerică* =0 atunci se generează numere aleatoare între 0 și 1. RNDM este de 2.5 ori mai rapid decît RND.

SCRN$ (FN K$)

Format:

SCRN$ (*linie* , *coloană*)

SCRN$ returnează caracterul-șir aflat în ecran în poziția (*linie, coloană*). *linie* este numărul liniei între 0 și 21, de sus în jos; *coloană* este numărul coloanei între 0 și 31, de la stînga la dreapta. *linie* și *coloană* pot constante. variabile sau expresii numerice eventual rotunjite la numărul întreg cel mai apropiat. SCRN$ recunoaște UDG.

SINE (FN S()

Format:

SINE (*expresie-numerică*)

SINE returnează sinusul unghiul exprimat în radiani cu o aproximație de numai 4 zecimale exacte, dar aproximativ de 6 ori mai rapid decît SIN.

STRING$ (FN S$)

Format:

STRING$ (*expresie-numerică* , *expresie-șir*)

STRING$ returnează un șir format prin repetarea *expresiei-șir* de un număr de ori dat de *expresie-numerică*.

TIME$

Format:

TIME$()

TIME$ returnează ca valoare șirul care reprezintă ora curentă în format CLOCK.

USING$ (FN U$)

Format:

USING$ (*format* , *expresie-numerică*)
unde format este #...#.#...# sau **0...0.0...0**
USING$ generează un şir care reprezintă un număr, conform formatului care stabileşte numărul de cifre înainte şi după . .
suprimă 0-urile (redundante) din faţa numărului
0 afişează 0-urile (redundante) din faţa şi coada numărului
XOR (FN X()
Format:
XOR (*expresie-numerică-1* , *expresie-numerică-2*)
XOR efectuează operaţia logică sau-exclusiv între *expresie-numerică-1* şi *expresie-numerică-2* care pot fi între 0 şi 65535.

3.4.4 Mesaje de eroare

Pe lîngă mesajele de eroare BASIC Spectrum, Beta BASIC introduce o serie de mesaje noi, în acelaşi format, conform comenzilor şi funcţiilor noi introduse.
S Missing LOOP
Lipseşte instrucţiunea LOOP
T I.OOP without DO
Lipseşte instrucţiunea DO
U No such line
Delete utilizat cu un număr de linie inexistent
V No POP data
Stiva GO SUB/DO-LOOP/PROG este vidă
W Missing DEF PROC
Lipseşte instrucţiunea DEF PROC
X No END PROC
Lipseşte instrucţiunea END PROC
Y Too hard
La execuţia comenzii RENUM a fost găsită o instrucţiune de salt la un număr de linie generat printr-o expresie numerică care nu poate fi evaluată.

3.4.5 Setul de caractere Beta BASIC

Cod	Comandă (G:UDG)	Cod	Comandă (G:UDG)
128	KEYWORDS	147	DO
129	DEF PROC	148	ELSE
130	PROC	149	FILL
131	END PROC	150	GET
132	RENUM	151	neutilizat
133	— —	152	EXIT IF
134	AUTO	153	WHILE

135	DELETE	154	UNTIL
136	— —	155	LOOP
137	JOIN	156	SORT
138	EDIT	157	ON ERROR
139	KEYIN	158	ON
140	— —	159	DPOKE
141	— —	160	POP
142	DEFKEY	161	ROLL
143	— —	162	SCROLL
144	ALTER	163	TRACE
145	*(BREAK-*)	164	USING
146	CLOCK		

4 PASCAL

4.1 INSTALAREA*

Alături de Forth, Pascal este cel mai eficient limbaj de programare pe calculatoarele compatibile Spectrum, fiind însă mai uşor de utilizat. După încărcarea cu **LOAD ""** compilatorul cere **Top of RAM?** adresa maximă RAM, **Top of RAM for T ?** adresa stivei pentru comanda editor T şi **Table size?** dimensiunea tabelei de simboluri utilizată de compilator.
Prin tastarea Enter ca răspuns la întrebări se consideră valori implicite: RAMTOP, -, memoria RAM liberă/16.
Din BASIC Spectrum în editorul compilatorului se întră prin:

RANDOMIZE USR 24598	şterge textul-sursă
RANDOMIZE USR 24603	păstrează textul-sursă
RANDOMIZE USR 24608	iniţializare

Taste funcţionale:

E	Enter:	Carriage return
CS1	Caps shift & 1:	ieşire din comanda-editor
CS0	Delete = Caps shift & 0:	şterge un caracter
CS8	Cursor = Caps shift & 8:	avans tab
CS5	Cursor = Caps shift & 5:	şterge toată linia

*HP4T15M este produs înregistrat al firmei HISOFT

4.2 EDITORUL

Promtul așteaptă o comandă de tipul:

comandă număr-1 , număr-2 , șir-1 , șir-2

unde *număr-1*, *număr-2* sînt numere întregi între 1 și 32767; *șir-1*, *șir-2* sînt șiruri de caractere de lungime între 0 și 20. Blank-ul nu este semnificativ decît în șiruri; , este separator

4.2.1 Generarea textului-sursă

I *număr-1 , număr-2*

Introduce numărătoarea automată a liniilor.
Valori implicite: *număr-1* = 10, *număr-2* = 10.
O linie text-sursă este de forma:

număr-linie secvență-program

Lungimea *secvenței-program* este de cel mult 80 de caractere, numărul de linie servește doar pentru facilitarea editării.
Orice linie nou introdusă, cu comanda I sau direct după prompt-ul >, se suprapune peste textul anterior astfel încît liniile anterioare cu același număr de linie se șterg.

4.2.2 Listare

L *număr-1 , număr-2*

Listează liniile textului-sursă avînd numerele între *număr-1* și *număr-2*; orice tastă efectuează scroll, cu excepția CS1.
Valori implicite: *număr-1* = 1, *număr-2* = 32767

K *număr*

Setează numărul de linii de listat înainte de scroll.
Valori implicite: *număr* = 16

4.2.3 Editare

D *număr-1 , număr-2*

Șterge liniile textului-sursă, avînd numerele între *număr-1* și *număr-2*.
Valori implicite: fără
Dacă *număr-1* > *număr-2* sau lipsesc argumente comanda este ignorată.

M *număr-1* , *număr-2*
Introduce linia *număr-1* la linia *număr-2*; linia *număr-1* rămîne în text; linia *număr-2* anterioară se şterge.
Valori implicite: fără

N *număr-1* , *număr-2*
Renumerotează textul-sursă începînd cu *număr-1* şi cu pasul *număr-2*
Valori implicite: fără

F *număr-1* , *număr-2* , *şir-1* , *şir-2*
şir-1 este căutat în liniile cu numerele între *număr-1* şi *număr-2*; dacă este găsit atunci se intră în editorul de linie (comanda E) cu cursorul pe primul caracter al *şir-1*; *şir-2* substituie *şir-1*.
Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără

E *număr*
Editor de linie.
Valori implicite: fără
Dacă număr lipseşte, comanda este ignorată.
Sub-comenzi:

blank	cursor dreapta
CS0	cursor stînga
CS8	avans tab
E	terminarea editării, cu modificări
C	supra-scriere, începînd de la poziţia curentă a cursorului
F	caută următorul *şir-1* din comanda F
I	inserează caractere la poziţia curentă a cursorului
K	şterge un caracter la poziţia cursorului
L	listează restul liniei în curs de editare
Q	terminarea editării, modificările sînt anulate
R	anulează modificările
S	înlocuieşte *şir-1* din comanda F
X	avansează cursorul la sfîrşitul liniei şi introduce sub-comanda I
Z	şterge toate caracterele de la cursor la sfîrşitul liniei

4.2.4 Comenzi de bandă magnetică

P *număr-1* , *număr-2* , *şir*
Salvează pe bandă magnetică liniile din textul-sursă între *număr-1* şi *număr-2*, într-un fişier cu numele *şir*.
Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără.

G , , *şir*
Încarcă de pe bandă magnetică fişierul (text-sursă) *şir* la sfîrşitul textului-sursă şi liniile sînt renumerotate.
Valori implicite: dacă *şir* lipseşte se încarcă primul fişier

T *număr*
Compilează textul-sursă începînd cu linia *număr* după care întreabă **OK?** şi la răspunsul **Y** codul-obiect direct executabil este salvat pe bandă magnetică cu nume (implicit) definit de comanda F. Codul poate fi apoi încărcat şi rulat din BASIC Spectrum cu **LOAD "" CODE : RANDOMIZE USR 24608**

W *număr-1* , *număr-2* , *şir*

Salvează liniile, avînd numărul între *număr-1* şi *număr-2*, cu numele *şir*, în format utilizabil de comanda de compilare **$F**.

4.2.5 Comenzi de uz general

B

Retur în BASIC Spectrum, reîntoarcere în Pascal cu:

RANDOMIZE USR 24598	şterge textul-sursă
RANDOMIZE USR 24603	păstrează textul-sursă
RANDOMIZE USR 24608	iniţializare

S , , *d*

Schimbă separatorul operanzilor din , în *d*; *d* nu poate fi blank.

V

Afişează bufferul.

4.2.6 Compilarea şi rularea programelor

C *număr*

Compilează programul începînd de la linia *număr*.
valoare implicită: *număr* = 1
La întîlnirea unei erori:

E introduce în editare linia curentă
P introduce în editare linia precedentă

O linie compilată are structura afişată:

adresă-linie-compilată *linie-compilată*

R

Rulează codul compilat fără erori de compilare.

4.3 STRUCTURI DE LIMBAJ

Setul de caractere

cifre: 0, .., 9; litere: a ... z , A ... Z
caractere speciale: + - * / = ^ < > ( ) [ ] . , : ; £
simboluri speciale:
operator de atribuire :=
operatori relaţionali <=, >=, <>
delimitatori de comentarii {, } sau (*, *)

Separatori

Spațiul, sfîrșitul de linie și comentariul sînt separatori. Se pot utiliza oricîți separatori și oriunde cu excepția utilizării lor în interiorul identificatorilor, numerelor și simbolurilor speciale. Acolo unde apar, caracterele speciale sînt de asemenea separatori, ; este separator de instrucțiuni care în cazuri speciale poate lipsi.

Cuvinte-cheie rezervate, identificatori predefiniți

ABS	**FALSE**	**ODD**	**SIN**
ADDR	**FOR**	**OF**	**SIZE**
AND	**FORWARD**	**ORD**	**SQR**
ARCTAN	**FRAC**	**OUT**	**SQRT**
ARRAY	**FUNCTION**	**PAGE**	**SUCC**
BEGIN	**GOTO**	**PACKED**	**TAN**
BOOLEAN	**HALT**	**PEEK**	**THEN**
CASE	**IF**	**POKE**	**TIN**
CHAR	**INCH**	**PRED**	**TO**
CHR	**INLINE**	**PROCEDURE**	**TOUT**
CONST	**INP**	**PROGRAM**	**TRUE**
COS	**INTEGER**	**RANDOM**	**TRUNC**
DIV	**LABEL**	**READ**	**TYPE**
DO	**LN**	**READLN**	**UNTIL**
DOWNTO	**MARK**	**REAL**	**USER**
ELSE	**MAXINT**	**RECORD**	**VAR**
END	**MOD**	**RELEASE**	**WHILE**
ENTIER	**NEW**	**REPEAT**	**WIDTH**
EOLN	**NIL**	**ROUND**	**WRITE**
EXP	**NOT**	**SET**	**WRITELN**

Identificatori (notație: id)

literă literă/cifră ...

Numai primele 10 caractere sînt semnificative. Se face distincție între litere majuscule și minuscule. Cuvintele-cheie trebuie scrise în majuscule, în editare sînt introduse literă cu literă dar în memorie sînt reținute pe un singur byte fiecare.

Întreg fără semn (notație: ifs)

cifră ...

Număr fără semn (notație: nfs)

ifs
ifs . ifs
ifs exp
ifs . ifs exp
£ *cifră-hex*

unde exp are formatul:

E *ifs*

Constantă fără semn (notație: cfs)

id-const
nfs
' *caracter* '
NIL

MAXINT
TRUE
FALSE

unde **NIL** este constantă de inițializare a pointerilori cu "nici o adresă"
MAXINT = 32767 cel mai mare întreg
TRUE, FALSE constante logice (adevărat, fals)
constantele acestea fiind predefinite.

Constantă (notație: const)

id-const
nfs
' *caracter* '
CHR (*const*)

Tip simplu (notație: tip-s)

INTEGER
REAL
CHAR
BOOLEAN
(*id* , ...)
const .. *const*

unde **INTEGER** tip întreg predefinit
REAL tip real predefinit
CHAR tip caracter predefinit
BOOLEAN tip logic predefinit

Un scalar enumarat poate avea maxim 256 elemente.
În tipul *const .. const* constantele trebuie să fie dintr-o mulțime de tip scalar enumerat ante/pre-definită.

Tip (notație: tip)

tip-s (a)
^*id-tip* (b)
ARRAY [*tip-s* , ...]**OF** *tip* (c)
SET OF *tip-s* (d)
RECORD *id* , ... : *tip* ; ... **END** (e)

unde: (b) utilizat pentru variabile dinamice
(c) definește tablouri de variabile *tip* cu indici (pentru acces) *tip-s*
(d) submulțimi
(e) variabile (tipuri) cu structuri complexe (cîmpuri)

PACKED poate fi folosit în fața tipurilor ARRAY, SET și RECORD dar acțiunea lui este nulă.

Variabile (notație: var)

id-var (a)
id-var [*expresie* , ...] (b)
id-var . *id-cîmp* (c)
id-cîmp (d)
id-cîmp [*expresie* , ...] (e)
id-cîmp . *id-cîmp* ... (f)

unde: (a) definire/acces la variabilele statice
(b) acces la elementele variabilelor tablou;
la accces/definirea tipului se pot utiliza în mod independent structurile [*expresie* ...] sau [*expresie*][...
(c) acces la cîmpurile variabilelor RECORD

(d) acces la cîmpurile variabilelor RECORD; în structuri WITH
(e) acces la cîmpurile tablou ale variabilelor RECORD
(f) acces la cîmpurile imbricate; în structuri WITH

Variabilele/cîmpurile dinamice sînt postfixate cu . .

Factor (notaţie: fact)

cfs	(a)
var	(b)
id-funcţie	(c)
id-funcţie (*expresie* , ...)	(d)
(*expresie*)	(e)
NOT *factor*	(f)
[*expresie*]	(g)
[*expresie* .. *expresie* , ...]	(h)

unde: (c),(d) reprezintă apeluri de funcţii cu sau fără parametri
(e) utilizarea parantezelor pentru inversarea ordinii de prioritate a operatorilor
(f) operaţie de negare logică sau aritmetică
(g),(h) mulţimi şi (sub)domenii

Termen (notaţie: termen)

factor
factor operator factor

unde operator este:

*	înmulţire pe numere întregi, reale; intersecţie pe mulţimi
/	împărţire pe numere reale
DIV	împărţire pe numere întregi
MOD	clase de resturi modulo
AND	şi logic /aritmetic

Expresie-simplă (notaţie: expr-s)

termen
termen operator termen ...
semn termen
semn termen operator termen ...

unde:

semn este + sau -

operator este:

+	adunare pe numere întregi, reale; reuniune pe mulţimi
-	scădere pe numere întregi, reale; diferenţă pe mulţimi
OR	sau logic /aritmetic

Expresie (notaţie: expresie)

expr-s
expr-s operator expr-s ...

unde operator este:

=	egalitate pe toate tipurile
< >	inegalitate pe toate tipurile
<	mai-mic pe tipuri simple, şiruri de caractere
>	mai-mare pe tipuri simple, şiruri de caractere
< =	mai-mic-egal pe tipuri simple, şiruri de caractere incluziune pe mulţimi
> =	mai-mare-egal pe tipuri simple, şiruri de caractere incluziune pe mulţimi
IN	apartenenţă pe mulţimi

Declaratii (notaţie: decl)

LABEL *ifs* , ... ; (a)
CONST *id* =*const* ; ... ; (b)
TYPE *id* =*tip* ; ... ; (c)
VAR *id* , ... : *tip* ; ... ; (d)

unde: (a) declară etichetele pentru GOTO, *ifs* număr de maxim 4 cifre
(b) defineşte constantele
(c) defineşte tipurile
(d) declară variabilele şi tipul lor

Declaraţiile sînt valabile în interiorul blocului în care apar, sînt opţionale şi ordinea lor este: LABEL, CONST, TYPE şi VAR.

INSTRUCŢIUNI SIMPLE ŞI STRUCTURATE (notaţie: instr-s)

Instrucţiunea de atribuire

id-var :=*expresie* (a)
id-func :=*expresie* (b)

unde (a) atribuire de valori variabilelor
(b) atribuire de valori funcţiilor; utilizată în blocul de definire al funcţiei respective

Instrucţiunea GOTO

GOTO *ifs*

Instrucţiune de salt necondiţionat. Eticheta este definită în acelaşi bloc şi saltul se poate face numai în acelaşi bloc la acelaşi nivel (nu se poate ieşi din FOR ... DO, funcţii sau proceduri).
Formatul etichetei este:

ifs : *instrucţiune*

Instrucţiunea IF-THEN

IF *expresie* **THEN** *instrucţiune*

Dacă *expresie* =**TRUE** atunci se execută *instrucţiune*; în orice situaţie se continuă cu instrucţiunile următoare.

Instrucţiunea IF-THEN-ELSE

IF *expresie* **THEN** *instrucţiune-1* **ELSE** *instrucţiune-2*

Dacă *expresie* =**TRUE** atunci se execută *instrucţiune-1* altfel se execută *instrucţiune-2*; în orice situaţie se continuă cu instrucţiunile următoare.

Instrucţiunea CASE-OF

CASE *expresie* **OF** *const* , ... : *instrucţiune* ; ... **END**

Dacă *expresie* are valoarea printre constante atunci se execută instrucţiunea corespunzătoare; în orice situaţie se continuă cu instrucţiunile de după END.
Nu este permis formatul vid: CASE OF END.

Instrucţiunea CASE-OF-ELSE

CASE *expresie* **OF** *const* , ... : *instrucţiune* ; ... **ELSE** *instrucţiune* **END**

Dacă *expresie* are valoarea printre constante atunci se execută instrucţiunea corespunzătoare; altfel se execută instrucţiunea de după ELSE; în orice situaţie se continuă cu instrucţiunile de după END.

Instrucţiunea WHILE-DO

WHILE *expresie* **DO** *instrucţiune*

Se efectuează *instrucţiune* atîta timp cît *expresie* =**TRUE**.

Instrucţiunea REPEAT-UNTIL

REPEAT *instrucțiune* ; ... **UNTIL** *expresie*

Se efectuează secvența de instrucțiuni pînă cînd *expresie* = **FALSE** dar cel puțin odată.

Instrucțiunea FOR-TO-DO

FOR *id-var* := *expresie-1* **TO** *expresie-2* **DO** *instrucțiune*

Buclă cu contor, la fiecare trecere *id-var* capătă valoarea următoare **SUCC** (*id-var*), *id-var* (contorul) ia valori între *expresie-1* și *expresie-2* inclusiv. Dacă *expresie-1* > *expresie-2* atunci instrucțiunea nu se efectuează. Contorul nu poate fi variabilă de tip RECORD.

Instrucțiunea FOR-DOWNTO-DO

FOR *id-var* := *expresie-1* **DOWNTO** *expresie-2* **DO** *instrucțiune*

Buclă cu contor, la fiecare trecere *id-var* capătă valoarea anterioară **PRED** (*id-var*), *id-var* (contorul) ia valori între *expresie-1* și *expresie-2* inclusiv. Dacă *expresie-1* < *expresie-2* atunci instrucțiunea nu se efectuează. Contorul nu poate fi variabilă de tip RECORD.

Instrucțiunea WITH-DO

WITH *id-var* , ... **DO** *instrucțiune*

Simplifică operațiile cu variabile-RECORD, în instrucțiune se folosesc numai identificatorii de cîmp. Nu sînt permise imbricările WITH.

Instrucțiunea vidă

; ;
BEGIN ;
; **END**
BEGIN BEGIN
BEGIN END

unde ; , BEGIN și END sînt delimitatorii instrucțiunii.

Instrucțiunea vidă poate fi etichetată

Apelul de procedură

id-procedură
id-procedură listă-parametri

Instrucțiune (notație: instrucțiune)

instr-s
BEGIN *instrucțiune* ; ... **END**

Listă de parametri (notație: list-param)

(*parametru*)
(*parametru* ; ...)

Parametru (notație: parametru)

id : *tip*
id , ... : *tip*
VAR *id* : *tip*
VAR *id* , ... : *tip*

Definiția funcțiilor (notație: def-func)

FUNCTION *id-func list-param* :*t ip* ; *bloc* ;

unde *tip* este tipul valorii (unice) returnate de funcție.

Funcțiile pot fi apelate recursiv. La apelarea unei funcții se face o corespondență biunivocă între lista parametrilor formali și lista parametrilor efectivi din apel și tipurile trebuie să corespundă.

Definiția procedurilor (notație: def-procedură)

PROCEDURE *id-procedură* ; *bloc* ;
PROCEDURE *id-procedură list-param* ; *bloc* ;

Procedurile nu pot fi utilizate în expresii. La apelarea unei proceduri se face o corespondenţă biunivocă între lista parametrilor formali şi lista parametrilor efectivi.

Declararea procedurilor (notaţie: decl-procedură)

PROCEDURE *id-procedură* ; **FORWARD**
PROCEDURE *id-procedură list-param* ; **FORWARD**

Se utilizează dacă se intenţionează utilizarea procedurilor înainte de definirea lor.

Bloc (notaţie: bloc)

decl-LABEL
decl-CONST
decl-TYPE
decl-VAR
decl-procedură
...
decl-procedură
def-func
...
def-func
def-procedură
...
def-procedură
BEGIN *instrucţiune* ; ... **END**

unde *decl-procedură*, *def-func* şi *def-procedură* pot fi utilizate în orice ordine permisă adică nu pot fi utilizate funcţiile nedefinite anterior sau proceduri nedefinite sau nedeclarate anterior; toate secvenţele sînt opţionale cu excepţia secvenţei **BEGIN** ... **END**.

Structura de program complet

PROGRAM *id* ; *bloc* .

unde instrucţiunile din cadrul blocului sînt instrucţiunile principale de execuţie.

4.4 Manipularea datelor în memorie

Întregi	stocaţi pe 2 bytes în complement faţă de 2
Reali	stocaţi pe 4 bytes în formă binară: bit31 = semn: 0 pozitiv/1 negativ bit30-bit8 = mantisa în formă normalizată: **1 .** ... bit7-bit0 = exponentul, întreg în complement faţă de 2 sînt stocaţi în regiştrii H, L, D, E iar în memorie în ordine inversă
Tablouri	nr-bytes = nr-elemente * nr-bytes-pe-element
Record-uri	se sumează nr-bytes pentru fiecare cîmp
Seturi	sînt stocate ca şiruri de biţi: nr-bytes = 1 + (nr–total–de–elemente–ale–multimii -1) DIV 8
Pointeri	ocupă 2 bytes fiecare, reprezenînd o adresă conform convenţiei

Intel

Variabilele globale

sînt declarate în programul principal şi li se alocă spaţiu în memorie pe stivă, de la adresa £8000 în jos, în ordinea declarării lor, atît spaţiu cît necesită fiecare

Variabilele locale

sînt declarate în sub-blocuri şi sînt stocate de la adresa IX+4 în jos, în ordinea declarării lor, atît spaţiu cît necesită fiecare; registrul IX este setat la începutul fiecărui bloc

Transmisia parametrilor şi valorilor de retur

sînt stocaţi de la adresa IX+2 în sus, în ordine inversă celei din list-param, astfel:

constantele sînt transmise prin valoare
variabilele sînt transmise prin referinţă (adresă: 2 bytes)
valorile de retur ale funcţiilor sînt puse la adresa cea mai mare

4.5 Funcţii şi proceduri predefinite

Parametri efectivi (notaţie: param-ef)

expresie

În expresii nu pot apărea proceduri sau funcţii (nu sînt valabile ca parametri).

4.5.1 Funcţii

FUNCŢII DE TRANSFORMARE A TIPULUI

CHR (*param-ef*)
parametrul este de tip întreg
funcţia returnează caracterul corespunzător codului ASCII; de tip char

ENTIER (*param-ef*)
parametrul este de tip întreg sau real
funcţia returnează întregul mai mic sau egal cu parametrul; de tip întreg

ORD (*param-ef*)
parametrul este de tip scalar dar nu real
funcţia returnează numărul de ordine al parametrului din setul căruia îi aparţine; de tip întreg

ROUND (*param-ef*)
parametrul este de tip întreg sau real
funcţia returnează parametrul rotunjit la întreg; de tip întreg

TRUNC (*param-ef*)
parametrul este de tip întreg sau real

funcţia returnează partea întreagă a parametrului; de tip întreg

FUNCŢII ARITMETICE

Parametrii sînt de tip întreg şi/sau real.

ABS (*param-ef*)
funcţia returnează valoarea absolută (modulul) parametrului; de acelaşi tip

ARCTAN (*param-ef*)
funcţia returnează arctangenta unghiului-parametru exprimat în radiani; de tip real

COS (*param-ef*)
funcţia returnează cosinusul unghiului-parametru exprimat în radiani; de tip real

EXP (*param-ef*)
funcţia returnează valoarea e^*parametru* (e=2.71828, numărul lui Euler); de tip real

FRAC (*param-ef*)
funcţia returnează *parametru* - **ENTIER** (*parametru*); de tip real

ODD (*param-ef*)
funcţia returnează **TRUE** dacă parametrul este impar, altfel **FALSE**; de tip logic

SIN (*param-ef*)
funcţia returnează sinusul unghiului-parametru exprimat în radiani; de tip real

SQR (*param-ef*)
funcţia returnează pătratul parametrului; de acelaşi tip

SQRT (*param-ef*)
funcţia returnează radicalul (rădăcina patrată) parametrului; de tip real

TAN (*param-ef*)
funcţia returnează tangenta unghiului-parametru exprimat în radiani; de tip real

FUNCŢII DE UZ GENERAL

ADDR (*id-var*)
funcţia returnează adresa de început a variabilei în memorie; de tip întreg

EOLN
funcţia returnează **TRUE** dacă următorul caracter citit este sfîrşit de linie (**EOL= CHR(13)**), altfel **FALSE**; de tip logic

INCH
funcţia citeşte tastatura şi returnează caracterul tastat sau altfel **CHR (0)**; de tip char

INP (*param-ef*)
parametrul este de tip întreg
se realizează instrucţiunile: **LD BC,** *param-ef*; **IN A, (C)** şi funcţia returnează caracterul citit; de tip char

PEEK (*param-ef*, *tip*)
parametrul este o adresă din memorie, de tip întreg
funcţia returnează date din memorie conform tip, care defineşte tipul funcţiei

PRED (*param-ef*)
parametrul este de tip scalar cu excepţia real
funcţia returnează predecesorul parametrului; de acelaşi tip

RANDOM
funcţia returnează un număr pseudo-aleator între 0 şi 255; de tip întreg

SIZE (*id-var*)

funcţia returnează numărul de bytes ocupaţi de variabilă în memorie; de tip întreg

SUCC (*param-ef*)

parametrul este de tip scalar cu excepţia real

funcţia returnează succesorul parametrului; de acelaşi tip

4.5.2 Proceduri

PROCEDURI DE SCRIERE/CITIRE

PAGE

Şterge ecranul respectiv, avansează imprimanta la începutul paginii următoare.

READ (*id-var*)

READ (*id-var*, ...)

Citeşte date de la buffer-ul de intrare al tastaturii şi le atribuie variabilelor în ordinea din lista. **EOLN=TRUE** la începutul programului.

READLN (*id-var*)

READLN (*id-var*, ...)

Analog READ dar citeşte într-un nou buffer de intrare al tastaturii.

WRITE (*param-ef*)

WRITE (*param-ef*, ...)

unde parametrul este de tipul:

expresie-1	(1)
expresie-1 : *expresie-2*	(2)
expresie-1 : *expresie-2* : *expresie-3*	(3)
expresie-1 : *expresie-2* : **H**	(4)

Scrie *expresie-1* la consolă, conform formatului:

- *expresie-1* de tip întreg: se utilizează (1) sau (2); *expresie-2* specifică numărul de cifre de afişat, dacă *expresie-1* are mai multe cifre atunci *expresie-2* este ignorată
- *expresie-1* de tip întreg hexazecimal: se utilizează (d); *expresie-2* specifică spaţiul/numărul de caractere alocat scrierii
- *expresie-1* de tip real: se utilizează (a), (b) sau (c)
- *expresie-3* reprezintă numărul de zecimale ale numărului de afişat în cazul în care *expresie-2* este dimensionată corespunzător
- *expresie-1* este de tip şir: se utilizează (a) sau (b), se afişează şirul, un caracter pe un spaţiu, cu blank-uri suplimentare dacă *expresie-2* o permite
- expresie-1 este de tip logic: se utilizează (a) sau (b), se afişează TRUE/FALSE, pe 4/5 spaţii, cu blank-uri suplimentare dacă expresie-2 o permite

Dacă se utilizează (a) sau *expresie-2* <=7 se afişează numărul în format standard. Dacă *expresie-2* este între 8 şi 12 atunci se afişează numărul în format standard cu 1 pînă la 5 zecimale după . .Dacă *expresie-2* >=13 se afişează blank-uri suplimentare. În general un număr real are nevoie de 12 spaţii-caracter pentru afişare. Cursorul (poziţia următoare de afişare) este pe prima poziţie liberă de după datele afişate.

WRITELN (*param-ef*)

WRITELN (*param-ef*, ...)

Analog WRITE dar cursorul (poziţia următoare de afişare) este pe linia următoare liberă de după datele afişate.

PROCEDURI DE MANIPULARE A VARIABILELOR DINAMICE

MARK (*id-var*)
Memorează într-o variabilă-pointer starea zonei de memorie Heap în care sînt stocate variabilele dinamice.

NEW (*id-var*)
Creează o variabilă dinamică a cărei adresă se află în variabilă.

RELEASE(*id-var*)
Eliberează zona de memorie Heap de variabilele dinamice conform ultimei proceduri MARK.

PROCEDURI DE UZ GENERAL

HALT
Opreşte rularea programului.

INLINE (*cod-byte* , ...)
Şirul de bytes reprezintă un program în cod-maşină Z80, care este inserat în programul compilat.

OUT (*param-ef-1* , *param-ef-2*)
param-ef-1 este de tip întreg, *param-ef-2* este de tip caracter
Se realizează instrucţiunile: **LD BC,** *param-ef-1*; **OUT (C), A.**

POKE (param-ef , id-var)
Stochează variabila în memorie, începînd de la adresa dată de parametru, care poate fi întreg în complement faţă de 2 sau hexazecimal. Variabila poate fi de orice tip cu excepţia seturilor.

TIN (*param-ef-1* , *param-ef-2*)
Încarcă de pe bandă magnetică un fişier al cărui nume este dat de *param-ef-1* (de tip şir) la adresa dată de *param-ef-2* (de tip întreg)

TOUT(*param-ef-1* , *param-ef-2* , *param-ef-3*)
Salvează pe bandă magnetică într-un fişier numit *param-ef-1* (şir de maxim 8 caractere) o zonă de memorie a cărei adresă de început este dată de *param-ef-2*, numărul de bytes (lungimea) fiind dat de *param-ef-3*.

USER (*param-ef*)
Apelează o subrutină în cod-maşină a cărei adresă de început este dată de parametrul de tip întreg, pentru valori mai decît 32767 se utilizează întregi negativi (întregii sînt memoraţi în complement faţă de 2). Subrutina trebuie să se termine cu instrucţiunea RET şi să nu afecteze registrul IX. Se pot utiliza şi constante hexazecimale.

4.6 Directive de compilare

Directivele de compilare se introduc în program separate de numărul de linie cu un blank.

$A +verifică ca indexul unei variabile-tablou să fie în domeniul declarat,

implicit +
$C +scanează tastatura în timpul execuţiei programului, implicit +
$F *şir* şirul trebuie să aibă 8 caractere (eventual blank-uri) şi să fie separat de $F cu un blank; comanda compilează de pe bandă magnetică un fişier program-sursă salvat cu comanda W
$I +verifică dacă există depăşiri la compararea a două numere întregi (depăşire există dacă între cele 2 numere este o diferenţă mai mare decît MAXINT (caz în care se poate genera rezultat incorect), implicit -
$L +listează liniile program-sursă la compilare, implicit +
$O +verifică depăşirile pentru adunările şi scăderile în numere întregi, implicit +
$P comută listarea pe imprimantă/ecran, implicit ecran
$S +verifică dacă există riscul depăşirii stivei la apelul unei proceduri sau funcţii, implicit +

4.7 Codurile erorilor

4.7.1 Erori la compilare

1 număr prea mare
2 lipseşte ;
3 identificator nedeclarat
4 lipseşte un identificator
5 se utilizează = nu :=
6 lipseşte =
7 instrucţiunea nu poate începe cu acest identificator
8 lipseşte :=
9 lipseşte)
10 tip eronat
11 lipseşte .
12 lipseşte un factor
13 lipseşte o constantă
14 identificatorul nu este o constantă
15 lipseşte **THEN**
16 lipseşte **DO**
17 lipseşte **TO/DOWNTO**
18 lipseşte (
19 tipul expresiei nu poate fi scris
20 lipseşte **OF**
21 lipseşte ,
22 lipseşte :
23 lipseşte **PROGRAM**
24 parametrul este de tip variabilă
25 lipseşte **BEGIN**

26 parametrul în **READ** este variabilă
27 nu pot fi comparate expresiile de tipul respectiv
28 tipul trebuie sa fie **INTEGER** sau **REAL**
29 acest tip de variabilă nu poate fi citit
30 nu este identificator de tip
31 lipseşte exponentul numărului real
32 lipseşte o expresie scalară
33 nu există şiruri nule (se utilizează CHR(0))
34 lipseşte [
35 lipseşte]
36 indexul tabloului trebuie să fie de tip scalar
37 lipseşte ..
38 lipseşte] sau ; în declaraţia tablourilor
39 marginea inferioară este mai mare decît marginea superioară
40 set prea mare (sînt permise maximum 256 elemente)
41 rezultatul funcţiei trebuie să fie identificator de tip
42 lipseşte] sau , în set
43 lipseşte], , sau .. în set
44 tipul parametrului trebuie să fie identificator de tip
45 primul factor într-o instrucţiune, cu excepţia asignării, nu poate fi un set nul
46 lipseşte un scalar, inclusiv real
47 lipseşte un scalar, exclusiv real
48 seturi incompatibile
49 seturile nu pot fi comparate cu < sau >
50 lipseşte **BEGIN**, **CONST**, **FORWARD**, **LABEL** sau **VAR**
51 lipseşte o cifră hexazecimală
52 seturile nu pot fi utilizate în **POKE**
53 tablou prea mare
54 lipseste : sau **END** în **RECORD**
55 lipseşte un identificator de cîmp
56 lipseşte variabila după **WITH**
57 variabila din **WITH** trebuie să fie de tip **RECORD**
58 lipseşte **WITH**
59 lipseşte întreg fără semn după **LABEL**
60 lipseşte întreg fără semn după **GOTO**
61 etichetă utilizată la un nivel eronat
62 etichetă nedeclarată
63 parametrul lui **SIZE** trebuie să fie o variabilă
64 pointerii pot fi comparaţi numai prin egalităţi
67 format incorect de scriere pentru întregi
68 şirurile nu pot conţine caracterul sfîrşit-de-linie (**EOL**)
69 parametrul lui **NEW**, **MARK** sau **RELEASE** trebuie să fie o variabilă de tip pointer
70 parametrul lui **ADDR** trebuie să fie o variabilă

4.7.2 Erori la execuţie

1 halt

2 depăşire
3 depăşirea memoriei **RAM**
4 împărţire cu 0
5 index prea mic
6 index prea mare
7 eroare matematică
8 număr prea mare
9 lipseşte un număr
10 linie prea lungă
11 lipseşte un exponent

prin compararea punctului de oprire unde a apărut eroarea cu adresele liniilor compilate se poate determina locul unde a apărut eroarea.

5 C

5.1 Instalarea*

C este un limbaj de programare de uz general, structurat care are un grad de portabilitate mai ridicat decît alte limbaje (cu excepţia Forth) şi al cărui compilator generează un cod mai eficient (rapid), datorită particularităţilor de limbaj. Deşi compilatorul este orientat pe calculatoarele de 8 biţi, care lucrează cu bandă magnetică, totuşi C nu îşi poate arăta întreaga sa forţă pe calculatoarele de tip Spectrum.
Încărcarea se face cu **LOAD** "" prin care (după încărcare) se lansează direct compilatorul.
Pentru a intra în editor se foloseşte Edit (CS1)
Pentru ca un program-sursă să fie compilat acesta trebuie să se termine cu EOF (end-of-file) (SSI).
Textul-sursă poate fi tastat direct în modul compilator, introdus din editor (memorie) cu comanda **# include** sau încărcat de pe bandă magnetică cu comanda **# include** *nume-fişier*.
Taste funcţionale:

Enter	NEW LINE (LINE FEED)	cod 10
CS1	(Caps Shift & 1):	Edit, afişat UDG 138, intrare în editor
SSI	(Symbol Shift & I):	EOF, cod -1, afişat UDG 134, sfîrşit de fişier
CS0	(Caps Shift & 0):	Delete, şterge un caracter

Intrările şi ieşirile, în C, se fac prin fişiere: bandă magnetică, memorie, ecran (fişier output), tastatură (fişier input).

*cc 1.1 este produs înregistrat al firmei HISOFT

5.2 Editorul

Prompt-ul > așteaptă o comandă de tipul:

comandă număr-1 , număr-2 , șir-1 , șir-2

unde *număr-1* , *număr-2* sînt numere întregi între 1 și 32767; *șir-1* , *șir-2* sînt șiruri de caractere de lungime între 0 și 20. Blank-ul nu este semnificativ decît în șiruri; , este separator. Comenzile pot fi introduse cu litere mari sau mici.

5.2.1 Generarea textului-sursă

I *număr-1* , *număr-2*

Introduce numărătoarea automată a liniilor.
Valori implicite: *număr-1* = 10, *număr-2* = 10.
O linie text-sursă este de forma:

număr-linie secvență-program

Lungimea *secvenței-program* este de cel mult 80 de caractere, numărul de linie servește doar pentru facilitarea editării.
Orice linie nou introdusă, cu comanda I sau direct după prompt-ul >, se suprapune peste textul anterior astfel încît liniile anterioare cu același număr de linie se șterg.

5.2.2 Listare

L *număr-1* , *număr-2*

Listează liniile textului-sursă avînd numerele între *număr-1* și *număr-2*; orice tastă efectuează scroll, cu excepția CS1.
Valori implicite: *număr-1* = 1, *număr-2* = 32767

K *număr*

Setează numărul de linii de listat înainte de scroll.
Valori implicite: *număr* = 16

W *număr-1* , *număr-2*

Listează la imprimantă liniile textului-sursă avînd numerele între *număr-1* și *număr-2*. Valori implicite: *număr-1* = 1, *număr-2* = 32767

5.2.3 Editare

D *număr-1* , *număr-2*

Sterge liniile textului-sursă, avînd numerele între *număr-1* şi *număr-2*.
Valori implicite: fără
Dacă *număr-1* >*număr-2* sau lipsesc argumente comanda este ignorată.

M *număr-1* , *număr-2*

Introduce linia *număr-1* la linia *număr-2*; linia *număr-1* rămîne în text; linia *număr-2* anterioară se şterge.
Valori implicite: fără

N *număr-1* , *număr-2*

Renumerotează textul-sursă începînd cu *număr-1* şi cu pasul *număr-2*
Valori implicite: fără

F *număr-1* , *număr-2* , *şir-1* , *şir-2*

şir-1 este căutat în liniile cu numerele între *număr-1* şi *număr-2*; dacă este găsit atunci se intră în editorul de linie (comanda E) cu cursorul pe primul caracter al *şir-1*; *şir-2* substituie *şir-1*.
Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără

E *număr*

Editor de linie.
Valori implicite: fără
Dacă *număr* lipseşte, comanda este ignorată.
Sub-comenzi:

blank	cursor dreapta
CS0	cursor stînga
CS8	avans tab
E	terminarea editării, cu modificări
C	supra-scriere, începînd de la poziţia curentă a cursorului
F	caută următorul *şir-1* din comanda F
I	inserează caractere la poziţia curentă a cursorului
K	şterge un caracter la poziţia cursorului
L	listează restul liniei în curs de editare
Q	terminarea editării, modificările sînt anulate
R	anulează modificările
S	înlocuieşte *şir-1* din comanda F
X	avansează cursorul la sfîrşitul liniei şi introduce sub-comanda I
Z	şterge toate caracterele de la cursor la sfîrşitul liniei

5.2.4 Comenzi de bandă magnetică

P *număr-1* , *număr-2* , *şir*

Salvează pe bandă magnetică liniile din textul-sursă între *număr-1* şi *număr-2*, într-un fişier cu numele *şir*.
Valori implicite: cele aflate în buffer la o utilizare anterioară, altfel fără.

G , , *şir*

Încarcă de pe bandă magnetică fişierul (text-sursă) *şir* la sfîrşitul textului-sursă şi liniile sînt renumerotate.
Valori implicite: dacă *şir* lipseşte se încarcă primul fişier

5.2.5 Comenzi de uz general

B

Retur în BASIC Spectrum, reîntoarcere în C cu: **RANDOMIZE USR 25200**

C

Retur în compilator.

S , , *d*

Schimbă separatorul operanzilor din , în *d* ; *d* nu poate fi blank.

V

Afişează buffer-ul.

5.3 Structuri de limbaj

5.3.1 Introducere

C este un limbaj cu grad înalt de portabilitate. Standardul C este definit în "The C Programming Language", Appendix A, autori: Brian W. Kenirghan şi Dennis M. Ritchie.

5.3.2 Convenţii lexicale

Tipuri de simboluri:

cuvinte-cheie	separatori
identificatori	constante
operatori	şiruri

Blank-urile, tabulatorii, NEW LINE şi comentariile (spaţiile albe) sînt ignorate la compilare.

Comentarii

/ text */*

Identificatori

literă literă/cifră

_ (sublinirea) este considerată literă; sînt semnificative numai primele 8 caractere; identificatorii externi sînt utilizaţi doar pentru declararea înainte a tipului funcţiilor

Cuvinte-cheie

Cuvintele-cheie nu pot fi utilizate în alte scopuri decît cele definite şi trebuie introduse în litere minuscule.

auto	**extern**	**static**
break	**for**	**struct**
case	**goto**	**switch**
cast	**if**	**typedef**
char	**int**	**union**
continue	**long**	**unsigned**
default	**register**	**void**
do	**return**	**while**
else	**short**	
entry	**sizeof**	

Constante (notație: const)

a) Întreg (int)

secvență-de-cifre

int este identic cu **long**

Dacă începe cu **0** se consideră număr octal, dacă începe cu **0x** sau **0X** se consideră număr hexazecimal.

b) Întreg lung (**long int**)

Dacă numărul se termină cu **l** sau **L** se consideră **long** (dar este identic cu **int**)

c) Caracter

'caracter'

Caractere speciale (pe un byte):

/n	New Line (Line Feed)
/t	Tab
/b	Back Space
/r	Carriage Return
/f	Form Feed
//	Back Slash
/'	Single Quote
/număr	bit-pattern
/0	Null

unde număr este octal pe 1-3 cifre

d) Numere în virgulă flotantă: neimplementat

Șiruri

"șir"

Caracteristici hard și de implementare

microprocesor Z80

set de caractere și coduri ASCII

char:	1byte	float:	neimplementat
int:	2bytes	double:	neimplementat
short:	2bytes	range:	-

5.3.3 Interpretarea identificatorilor

Un identificator-variabilă are 2 caracteristici:

clasa de memorie

tipul

Clasele de memorie sînt:

auto: sînt valabile doar în interiorul blocului în care au fost definite şi valorile nu sînt reţinute între 2 apeluri succesive

static: sînt valabile doar în interiorul blocului în care au fost definite şi valorile sînt reţinute între 2 apeluri succesive

extern: variabilele sînt globale, putînd fi apelate de orice funcţie

register: recunoscut dar tratat ca auto (din motive hard)

Tipurile de bază sînt:

char: tip caracter

int: tip întreg (pe 2 bytes) cu sub-tipurile:
short int, int, long int (manipulate toate la fel)
unsigned int: întreg fără semn

Cu tipurile de bază se pot construi:
tablouri cu componente de acelaşi tip
funcţii cu rezultat de orice tip
adrese ale datelor de orice tip
structuri (**struct**) cu componente diferite
uniuni (**union**) de componente diferite

În general metodele de construire a unor noi tipuri se pot folosi succesiv şi recursiv.

5.3.4 Obiecte şi s-valori (l-values)

Obiectele sînt regiuni de memorie manipulabile.
S-valorile (*l-values*) sînt expresii care se află în partea stîngă a unei atribuiri şi sînt operanzi de tip referintă, adică:
identificator de variabilă simplă
element de tablou (variabilă indexată)
element de structură desemnat prin numele structurii
element de structură desemnat prin adresa structurii
pointer

5.3.5 Conversii

Caractere şi întregi

Caracterele şi întregii pot fi utilizaţi unul în locul celuilalt fără restricţii. Caracterele apar ca întregi între 0 şi 255 (fără extinderea semnului).

Adrese (pointeri şi întregi)

Adresele sînt tratate ca întregi fără semn, dar rezultatul scăderii a două adrese este un întreg cu semn.

Întregi fără semn

Dacă într-o expresie există un întreg fără semn, atunci ceilalţi întregi sînt convertiţi în întregi fără semn, iar rezultatul este de tip întreg fără semn.

Conversii aritmetice
Operanzii de tip caracter sînt transformaţi în tip int.
Dacă există operanzi de tip întreg fără semn, atunci toţi ceilalţi sînt convertiţi la acest tip şi rezultatul este de acest tip.
Altfel rezultatul este de tip întreg (tipurile **short** şi **long** sînt la fel ca **int**).

5.3.6 Expresii (notaţie: expr)

Expresiile sînt formate din operanzi şi operatori:
operanzii sînt valori sau adrese de memorie
operatorii se aplică operanzilor
rezultatul evaluării unei expresii este o valoare sau o adresă
Ordinea de aplicare a operatorilor depinde de prioritatea asociată fiecărui operator şi de prezenţa parantezelor utilizate pentru inversarea ordinii de prioritate/aplicare
În fiecare sub-paragraf operatorii au acelaşi grad de prioritate. Sub-paragrafele sînt ordonate în ordinea descreşterii gradului de prioritate. Altfel ordinea de evaluare a unei expresii nu este definită.
Depăşirile întregi şi împărţirile cu 0 sînt ignorate.
Expresii primare (notaţie: expr-p)

id	(a)
const	(b)
şir	(c)
(expr)	(d)
expr-p [*expr*] ... [*expr*]	(e)
expr-p ()	(f)
expr-p (*expr*)	(g)
expr-p (*expr* ... , *expr*)	(h)
s-valoare . *id*	(i)
expr-p -> *id*	(j)

unde:

(a) *id* are tipul definit în declaraţii, un *id* declarat ca tablou este de tip pointer şi are asociată adresa primului element din tablou
(b) constanta tip **int**, caracterele au tipul **int**
(c) are ca valoare adresa primului caracter din şir (şirul este asimilat cu un tablou de caractere)
(d) utilizarea parantezelor pentru marcarea ordinii de evaluare, parantezele nu modifică interpretarea *s-valorilor*
(e) utilizarea variabilelor indexate (element de tablou) unde *expr-p* este numele tabloului şi *expr* este indexul;
expr-p [*expr*] este identică cu *((*expr-p*)+(*expr*))
(f),(g),(h) apeluri de funcţii ale căror parametrii sînt *expr* , ...; parametrii funcţiilor sînt transmişi prin valoare pe stiva de parametri, **char** este convertit la **int**, tablourile sînt transmise prin pointeri; sînt permise apelurile recursive ale oricăror funcţii, funcţiile returnează rezultatele conform tipului definit sau (implicit) **int**
(i) utilizarea elementelor variabilelor de tip **struct** sau **union**
(j) *expr-p* trebuie să fie un pointer către o variabilă **struct** sau **union** şi

id trebuie să fie un identificator al unui membru al variabilei; rezultatul este s-valoarea pointer referitoare la membrul respectiv
expr-p este identică cu (**expr-p*) . *id*

Operatorii din expresiile primare () [ ], -> au prioritate maximă față de ceilalți operatori și se evaluează de la stînga la dreapta.

Expresii unare (notație: expr-u)

* *expr*	(a)
& *s-valoare*	(b)
- *expr*	(c)
! *expr*	(d)
~ *expr*	(e)
+ + *s-valoare*	(f)
- - *s-valoare*	(g)
s-valoare + +	(h)
s-valoare - -	(i)
cast (*nume-tip*) *expresie*	(j)
size of *expr*	(k)
size of (*nume-tip*)	(l)

unde:

(a) * este operator de indirectare; *expr* este un pointer, rezultatul este o *s-valoare*

(b) rezultatul este un pointer către obiectul referit de *s-valoare*

(c) - schimbă semnul operatorului pe care îl precede

(d) ! este operator de negație logică:
expr = 0 => !*expr* = 1
expr < >0 => !*expr* = 0

(e) ~ este operator de negare pe biți (complement față de 1)

(f),(h) + + este operator de incrementare a *s-valorii*
(f) incrementează *s-valoarea* apoi produce valoarea incrementată
+ +*s-valoare* este identică cu *s-valoare* + = 1
(h) produce *s-valoarea* apoi o incrementează
tipul rezultatului este dat de tipul *s-valorii*

(g),(i) - - este operator de decrementare a *s-valorii*
(g) decrementează *s-valoarea* apoi produce valoarea decrementată
- -*s-valoare* este identică cu *s-valoare* - = 1
(i) produce *s-valoarea* apoi o decrementează
tipul rezultatului este dat de tipul *s-valorii*

(j) forțează conversia rezultatului evaluării expresiei la *nume-tip* (predefinit sau definit anterior)

(k) returnează dimensiunea operandului în bytes

(l) returnează dimensiunea unui obiect de categoria *nume-tip* (predefinit sau definit anterior)

Operatori multiplicativi

expr * *expr*
expr / *expr*
expr % *expr*

unde: * este operator de înmulțire
/ este operator de împărțire
% este operator modulo (rezultatul este restul împărțirii primei *expr* la

a 2-a)
Operatorii se aplică de la stînga la dreapta.
Operatori aditivi
expr + *expr*
expr - *expr*
unde: + expr este operator de adunare
/ expr este operator de scădere
Operatorii se aplică de la stînga la dreapta.
Adunarea/scăderea unui întreg la un pointer se face după înmulţirea întregului cu lungimea valorii adresate de pointer, iar rezultatul este un pointer.
Scăderea a 2 pointeri, asupra aceluiaşi tablou returnează ca valoare numărul de obiecte aflate între cei 2 pointeri (la evaluare se împarte rezultatul preliminar cu lungimea obiectului.

Operatori de shift

expr-1 < < *expr-2*
expr-1 > > *expr-2*

unde < < şi > > realizează deplasarea la stînga (< <)/dreapta (> >) a *expr-1* tratat ca o configuraţie binară cu un număr de poziţii egal cu *expr-2*. Deplasarea se face cu introducerea de 0-uri printr-o extremitate şi pierderea cifrelor binare care ies prin cealaltă extremitate.
> > realizează shift aritmetic dacă *expr-1* este de tip int (cu excepţia **unsigned**).

Operatori relaţionali

expr-1 < *expr-2*
expr-1 > *expr-2*
expr-1 < = *expr-2*
expr-1 > = *expr-2*

unde: < mai mic < = mai mic sau egal
> mai mare > = mai mare sau egal

Operatorii se aplică de la stînga la dreapta.
Rezultatul este de tip **int**: 0 pentru fals,1 pentru adevărat.
2 pointeri pot fi comparaţi, rezultatul fiind semnificativ doar dacă sînt în cadrul aceluiaşi tablou.

Operatori de egalitate

expr-1 = = *expr-2*
expr-1 != *expr-2*

unde: = = egal cu != nu este egal cu
Operatorii se aplică de la stînga la dreapta.
Rezultatul este de tip **int**: 0 pentru fals, 1 pentru adevărat.
2 pointeri pot fi comparaţi, rezultatul fiind semnificativ doar dacă sînt în cadrul aceluiaşi tablou. Un pointer egal cu 0 semnifică că acesta nu adresează nimic.

Operatorul binar şi

expr & *expr*

& realizează şi logic pe biţi; este operator asociativ.
Operatorul binar sau-exclusiv

expr ^ *expr*

realizează sau-exclusiv logic pe biţi; este operator asociativ.

Operatorul binar sau

expr | *expr*

| realizează sau logic pe biţi; este operator asociativ.

Operatorul logic şi

expr && *expr*

&& realizează şi logic; se aplică de la stînga la dreapta
Rezultatul este de tip **int**: 0 dacă una din *expr* este nulă
1 dacă ambele *expr* sînt nenule
Operatorii pot fi de tipuri diferite, dar predefinite sau pointeri.

Operatorul logic sau

expr | | *expr*

realizează sau logic; se aplică de la stînga la dreapta
Rezultatul este de tip **int**: 0 dacă ambele *expr* sînt nule
1 dacă una din *expr* este nenulă
Operatorii pot fi de tipuri diferite, dar predefinite sau pointeri.

Operatorul condiţional

expr-1 ? *expr-2* : *expr-3*

Se aplică de la dreapta la stînga.
Dacă *expr-1* este nenulă , rezultatul este valoarea data de *expr-2* altfel rezultatul este *expr-3.*

Operatori de atribuire

s-valoare =*expr* (a)
s-valoare op =*expr* (b)

unde *op* este operator care poate fi: +, -, *, /, %, > >, < <, ^,!
Atribuirea se aplică de la dreapta la stînga.
(a) reprezintă operaţia de atribuire, *s-valoarea* capătă valoarea *expr*
(b) realizează *s-valoare* =*s-valoare op* (*expr*); pentru + sau - *s-valorile* pot fi pointeri
Rezultatul este de tipul *s-valorii.* Nu este permisă atribuirea între pointeri şi întregi cu excepţia **0** care înseamnă nici o adresă.

5.3.7 Declaraţii (notaţie: decl)

specificatori-decl ;
specificatori-decl listă-decl ;

unde *specificatori-decl* este:

specificator-tip
specificator-tip specificatori-decl
specificator-cm
specificator-cm specificatori-decl

specificatori-decl reprezintă o secvenţă de tipuri şi clase de memorare.
lista-decl conţine identificatorii de declarat.

Clase de memorare (notaţie: cm)

auto
static
extern
register
tipedef

extern este utilizat doar pentru funcţiile din bibliotecă, care nu sînt de tip **int**.

Specificatori de tip (notaţie: specificator-tip)

char
short
int
long
unsigned
specificator-union-sau-struct
specificator-typedef

int poate fi precedat de **short**, **long** sau **unsigned**.

Lista declaratorilor (notaţie: lista-decl)

declarator-i
declarator-i , ... *declarator-i*

unde *declarator-i* este:

declarator
declarator iniţializator

iniţializator realizează iniţializarea declaratorilor şi *declarator* este:

id (a)
(*declarator*) (b)
* *declarator* (c)
declarator ( ) (d)
declarator [ ] (e)
declarator [*expr-const*] (f)
declarator ... (g)

Obiectele declarate sînt variabile valorice (a), pointeri (c), funcţii (d) sau tablouri (e, f, g).

Funcţiile nu pot returna pointeri către acestea, nu există tablouri de funcţii, dar pot exista tablouri de pointeri către funcţii.

Structuri şi uniuni (notaţie: specificator-union-sau-struct)

struct-sau-union {*listă-decl-struct* }
struct-sau-union id {*listă-decl-struct* }
struct-sau-union id

unde *struct-sau-union* este:

struct
union

şi *listă-decl-struct* este:

declaraţie-struct
declaraţie-struct ... *declaraţie-struct*

unde *declaraţie-struct* este:

specificator-tip listă-declaratori-struct

unde *listă-declatori-struct* este:

declarator
declarator ..., *declarator*

Structura este o variabilă care poate conţine mai mulţi membri de tipuri diferite, sînt permise cîmpuri vide (neocupate de membri), adresele obiectelor declarate sînt în ordine crescătoare conform citirii de la stînga la dreapta.

Uniunea este o variabilă care la un moment dat poate conţine doar un membru al unei colecţii, membrii putînd fi de tipuri diferite.

Structurile sau uniunile pot fi autoreferite numai prin pointeri. Identificatorii variabilelor, structurilor, uniunilor şi membrilor trebuie să fie diferite.

Iniţializarea (notaţie: iniţializator)

=*expr-const* (a)

= {*expr-const*... , *expr-const* } (b)

unde: (a) inițializează variabilele scalare
(b) inițializează variabilele tablou și **struct**

Variabilele **static** și **external** neinițializate explicit sînt inițializate cu **0**. Variabilele **auto** nu pot fi inițializate.

Nume de tipuri (notație: nume-tip)

Un tip poate fi definit prin:

typedef tip id

unde: *tip* este un tip predefinit sau construcții **struct/union**
id este utilizat ca specificator de tip

5.3.8 Instrucțiuni

Instrucțiunile sînt executate secvențial cu excepția cazurilor menționate explicit

Expresii

expresie ;

În general expresia este de atribuire sau apel de funcție.

Instrucțiunea compusă/bloc (notație: bloc)

{*listă-decl listă-instrucțiuni* }

unde *listă-decl* este:

/* *nimic* */
decl
decl ; ... *decl* ;

și *listă-instructiuni* este:

instrucțiune ;
instrucțiune ; ... *instructiune* ;

Instrucțiunea if

if (*expr*) *instrucțiune-1*
if (*expr*) *instrucțiune-1* **else** *instrucțiune-2*

Dacă *expr* este diferită de 0 (adevarată) atunci se execută *instrucțiune-1* altfel se executa *instrucțiune-2* (dacă există). În instrucțiunile **if** imbricate **else** aparține instrucțiunii **if** celei mai apropiate anterior definite.

Instrucțiunea while

while (*expr*) *instrucțiune*

Instrucțiunea este executată atîta timp cît expresie este diferită de 0 (adevărată), testul se efectuează înaintea execuției instrucțiunii.

Instrucțiunea do-while

do *instrucțiune* **while** (expr)

Instrucțiunea este executată pînă cînd *expr* devine **0** (fals), testul se efectuează după execuția instrucțiunii.

Instrucțiunea for

for (*expr-1* ; *expr-2* ; *expr-3*) *instrucțiune*

reprezintă bucla cu contor, unde:
expr-1 reprezintă inițializarea contorului
expr-2 reprezintă testul de sfîrșit al buclei
expr-3 reprezintă actualizarea contorului

Instrucţiunea este echivalentă cu

expr-1
while (*expr-2*) {*instrucţiune expr-3* ; }

expr-1/2/3 sînt opţionale, dar separatorii ; sînt obligatorii.
Valori implicite: *expr-2* = 1 (adevărat, bucla infinită)

Instrucţiunea switch

switch (*expr*)
{**case** *expr-const* : ... *expr-const* : *instrucţiune*
...
case *expr-const* : ... *expr-const* : *instrucţiune*
default : *instrucţiune* }

Se evaluează *expr* şi se execută salt la prima instrucţiune etichetată **case** pentru care *expr* este egală cu *expr-const* sau la **default** în caz contrar, dacă există,şi se execută în continuare toate instrucţiunile care urmează. Poate exista o singură etichetă, opţională, **default**; pot exista mai multe etichete **case** ataşate unei instrucţiuni.

Instrucţiunea break

break ;

Forţează salt la sfîrşitul instrucţiunilor: **while**, **for** sau **switch** în care se găseşte (imediat superioare).

Instrucţiunea continue

continue ;

Forţează salt la sfîrşitul corpului de instrucţiuni ale buclelor **while**, **do-while** sau **for** în care se găseşte (imediat superioare) fără părăsirea acestora.

Instrucţiunea return

return ; (a)
return *expr* ; (b)

este opţională, se consideră implicit (a); specifică valoarea returnată de funcţie unde:

(a) valoarea returnată este nedefinită (nespecificată)
(b) *expr* este valoarea returnată de funcţie, care este convertită dacă este cazul la tipul funcţiei

Instrucţiunea goto

goto *id-eticheta*

generează salt necondiţionat la instrucţiunea etichetată, *id* este valabil în funcţia curentă.

Etichetarea instrucţiunilor

id-etichetă : *instrucţiune*

unde *id-etichetă* reprezintă eticheta instrucţiunii, utilizată de **goto**.

Instrucţiunea vidă

;

efectul instrucţiunii vide este nul, instrucţiunea vidă poate fi etichetată.

Instrucţiunea inline

inline (*expr-const*)
inline (*expr-const* ... , *expr-const*)

permite introducerea unui program în cod-maşină Z80
Dacă *expr-const* este între 0 şi 255 atunci este evaluată la 1 byte altfel este evaluată la 2 bytes.

5.3.9 Definiţii externe

Un program C este compus dintr-o serie de definiţii externe printre care se află o funcţie program-principal denumită **main** care este apelată la începutul rulări programului.
Definiţiile externe au clasa de memorie **extern** implicit sau **static** şi sînt valabile în tot fişierul-program.
Definiţiile externe au acelaşi format ca toate celelalte definiţii cu excepţia faptului ca la acest nivel pot fi definite funcţiile.
Definiţiile funcţiilor (notaţie: func)

specificator-decl declator-func corp-func
declator-func corp-func

unde *declator-func* este:

declator () sau
declator (*id*) sau
declator (*id* ... , *id*)

şi *corp-func* este:

listă-decl bloc

Specificatorii claselor de memorie permişi sînt **extern** şi **static** cu semnificaţie şi utilizare normală, **auto** plasat între *declator-func* şi *corp-func* permite definirea şi utilizarea funcţiilor cu număr variabil de parametri. Compilatorul plasează argumentele efective ale funcţiei (la apelul ei) după celelalte argumente ca o secventa de bytes. Această secvenţă este utilizată de funcţie ca argument al ei.
lista-decl conţine declaraţii referitoare doar la parametrii funcţiei.
Definiţiile datelor

decl

clasele de memorizare ale datelor pot fi doar **extern** (implicit) sau **static**.

5.3.10 Domeniile de valabilitate ale declaraţiilor

Întregul program-sursă este compilat odată
Domeniul de valabilitate a argumentelor formale ale funcţiilor este în blocul de defiţitie al funcţiei respective
Domeniul de valabilitate lexical al unei variabile externe declarate în afara funcţiilor este programul-sursă
Funcţia **main** este definită fără argumente

5.3.11 Preprocesorul

Compilatorul C conţine un preprocesor care caută în programul-sursă şi interpretează toate liniile care încep cu caracterul # .
Aceste linii de control ale compilării se numesc directive preprocesor

Directiva define

 # define *id şir*

înlocuieşte în programul-sursă *id* cu *şir*

Directiva include

 # include (a)
 # include " *nume-fişier* " (b)
 # include < *nume-fişier* > (c)
 # include *nume-fişier* (d)
 # include ?*nume-fişier* **?** (c)

unde:

(a) compilează programul-sursă din editor
(b),(c),(d) încarcă şi compilează un text-sursă fişier de pe bandă magnetică cu numele *nume-fişier*
(e) în textul-sursă, fişier de pe bandă magnetică cu *nume-fişier*, sînt căutate, încărcate şi compilate doar funcţiile utilizate în program. În fişier funcţiile care apelează alte funcţii trebuie puse la început pentru evitarea recitirii fişierului.

Directiva list

 # list

unde + activează / - dezactivează listarea liniilor următoare ale textului-sursă în decursul compilării.

Directiva direct

 # direct

unde + activează / - dezactivează execuţia directă a instrucţiunilor (modul interpretor)

Directiva translate

 # translate *nume-fişier*

salvează pe bandă magnetică codul compilat într-un fişier cu numele *nume-fişier*. Programul-obiect poate fi încărcat şi executat cu secvenţa:

LOAD CODE "" : RANDOMIZE USR 25200

Directiva error

 # error

şterge mesajele de eroare, eliberînd memoria pentru programele mari

5.3.12 Expresii constante (notaţie: expr-const)

Expresiile evaluate la o constantă implică utilizarea constantelor **int** şi **char**, expresii **sizeof**, operatorii unari: **-**, **~**; operatorii binari: **+**, *****, **/**, **%**, **&**, **|**, **^**, **< <**, **> >**, **= =**, **!=**, **<**, **>**, **< =**, **> =** , operatorul ternar **? :** şi parantezele **( )** pentru definirea ordinii de evaluare a operatorilor.

5.4 Funcţii predefinite

Funcţiile predefinite sînt compilate şi introduse automat în program.

int fclose (*fp*) **FILE** **fp* ;

Închide fişierul (deschis pentru scriere/citire de ***fopen**) indicat de pointerul *fp* şi eliberează buffer-ul (pentru un alt fişier).

FILE *fopen (*name* , *mode*) **char** **name* , **mode* ;

Deschide un fişier denumit de şirul *name* pentru operaţia definită de şirul *mode* care poate fi: **r** pentru citire, **w** pentru scriere; are ca rezultat un pointer către structura de descriere a fişierului utilizat de alte funcţii, pointerul este 0 în caz de eroare.

void fprintf (*fp* , *control* , *arg1* , *arg2* .. .) **FILE** **fp* ; **char** **control* ;

Scrie în fişierul indicat de pointerul *fp* conform şirului *control* care poate conţine caractere normale de scris (şiruri) (inclusiv caractere speciale) şi /sau specificatori de format:

% d întreg zecimal
% o octal fără semn
% x hexazecimal fără semn
% u întreg fără semn
% c caracter
% s şir terminat cu caracterul **\0**
% % scrie caracterul %

Între % şi caracter pot fi modificatori care semnifică:

- aliniere la stînga (implicit la dreapta)
0 completare cu 0
şir-de-cifre specificator de lăţime minimă cîmp
. *şir-de-cifre* specificator de lăţime maximă cîmp
L pentru **long int**, acceptat dar ignorat

(se pot utiliza mai mulţi modificatori în acelaşi specificator)

int getc (*fp*) **FILE** **fp* ;

Citeşte următorul caracter din fişierul indicat de pointerul *fp*, returnează -1 pentru EOF.

int getchar ( ) ;

Citeşte un caracter de la tastatură într-un buffer pînă la apăsarea Enter. Se poate utiliza CS0.

int isalpha (*c*) **char**(*c*) ;

Returnează 1 (adevărat) dacă *c* este literă, altfel 0 (fals).

int isdigit(*c*) **char**(*c*) ;

Returnează 1 (adevărat) dacă *c* este cifră, altfel 0 (fals).

int islower(*c*) **char**(*c*) ;

Returnează 1 (adevărat) dacă *c* este literă minusculă, altfel 0 (fals).

int isspace (*c*) **char**(*c*) ;

Returnează 1 (adevărat) dacă *c* este blank, new-line sau tab, altfel 0 (fals).

int isupper(*c*) **char**(*c*) ;

Returnează 1 (adevărat) dacă *c* este literă majusculă, altfel 0 (fals).

int keyhit( ) ;

Returnează 1 (adevărat) dacă a fost apăsată o tastă, altfel 0 (fals); trebuie resetată înainte de o reutilizare prin resetarea bitului 5 al variabilei sistem FLAGS (utilizează rutina 028E).

void move (*dest* , *source* , *length*)
char **dest* , **source* ; *unsigned lenght* ;

Mută un şir de bytes, de lungime *length* şi care începe la *source*, începînd de la *dest*. Copierea se face înainte de o eventuală suprascriere (fără pierderea informaţiei).

int putc (*c* , *fp*) **int** *c* ; **FILE** **fp* ;

Scrie caracterul *c* în fişierul indicat de pointerul *fp* şi returnează caracterul ca rezultat.

void printf (*control* , *arg1* , *arg2* ...) **char** **control* ;

Scrie, conform şirului *control*, pe ecran: *arg1*, *arg2* ... Şirul *control* are aceleaşi caracteristici cu cel utilizat de funcţia **fprintf**.

int putchar (*c*) **int** *c* ;

Scrie caracterul *c* pe ecran şi returnează caracterul ca rezultat.

int rawin ( )

Citeşte un caracter de la tastatură fără prelucrarea codurilor. Bitul 5 al variabilei-sistem FLAGS este setat la apăsarea tastei, citeşte variabila-sistem LAST-K şi resetează fanionul.

void sprintf (*s* , *control* , *arg1* , *arg2* ...) **char** **s* , **control* ;

Scrie, conform şirului *control*, în şirul *s*: *arg1*, *arg2*... Şirul *control* are aceleaşi caracteristici ca cel utilizat de funcţia **fprintf**.

void swap (*p* , *q* , *lenght*) **char** **p* , **q* ; **unsigned** *lenght* ;

Inversează 2 zone de memorie avînd numărul de bytes *lenght* fiecare şi pointerii *p* şi *q*.

char tolower (c) **char** *c* ;

Returnează caracterul literă minusculă dacă *c* a fost literă majusculă, altfel *c* nu este modificat.

char toupper(*c*) **char** *c* ;

Returnează caracterul literă majusculă dacă *c* a fost literă minusculă, altfel *c* nu este modificat.

int ungetc (*c* , *fp*) **int** *c* , **FILE** **fp* ;

Pune caracterul *c* înapoi în fişierul *fp*.

5.5 Codurile erorilor

0 lipseşte caracter
1 numerele în virgulă flotantă nu sînt implementate
2 constantă caracter eronată
3 nu este comandă preprocesor
4 buffer-ul macro plin
5 doar identificatorii pot fi definiţi ca macro
6 parametrii macro nu sînt implementaţi
7 fişierul nu poate fi deschis

8 # **include** imbricate nu sînt implementate
9 lipseşte **while**
10 **break** poate fi utilizat doar în **do-loop**, **while**, **for** sau **swich**
11 **continue** poate fi utilizat doar în **do-loop**, **while** sau **for**
12 **case** sau **default** pot fi utilizate doar în **switch**
13 pot fi utilizate maxim 50 etichete **case** într-o instrucţiune **switch**
14 poate fi utilizată doar o etichetă **default** într-o instrucţiune **switch**
15 lipseşte eticheta în **goto**
16 etichetă utilizată de mai multe ori (în **goto** sau ca etichetă)
17 **#direct** nu poate fi utilizat cu **#translate**
18 tabela de variabile utilizată la compilare este plină (prea multe variabile globale şi/sau funcţii cu prea multe variabile)
19 tabela de tipuri utilizată la compilare este plină
20 declaraţii multiple pentru acelaşi identificator, tip eronat
21 declaraţii multiple pentru acelaşi identificator, clasa de memorie eronată
22 tabela de variabile globale utilizate la compilare este plină
23 zona de date globale este plină (variabile globale prea multe şi/sau cu dimensiuni prea mari)
24 declaraţie multiplă pentru acelaşi identificator
25 tabela de variabile locale utilizată la compilare este plină
26 parametrul efectiv nu este în lista parametrilor formali a funcţiei apelate
27 variabilă nedefinită
28 funcţia nu poate returna acest tip
29 tablourile de funcţii nu sînt permise
30 prea multe paranteze şi/sau argumente în expresie
31 prea mulţi operatori în expresie
32 tipurile variabilelor sînt incompatibile în expresie
33 tipul operandului sau operatorul eronat în expresie
34 lipseşte o *s-valoare* (*l-value*)
35 nu este membru al unei structuri sau uniuni
36 lipseşte o expresie primară
37 variabilă nedefinită
38 lipseşte un nume de tip
39 lipseşte o expresie constantă
40 expresia poate fi folosită doar ca apel de funcţie
41 caracterul : utilizat incorect
42 lipseşte o *s-valoare* (*l-value*) la operatorul de atribuire
43 lipseşte : la operatorul ternar ?
44 lipseşte un pointer
45 tipul parametrului funcţiei este incorect
46 numerele în virgulă flotantă nu sînt implementate
47 operatorul nu funcţionează cu numere în virgulă flotantă
48 declaraţie eronată
49 clasa de memorie eronată în context
51 identificator utilizat multiplu în structuri sau uniuni
52 structura utilizată ca parametru al funcţiei trebuie declarată
53 o structura nu se poate contine pe sine
54 declarator eronat
55 lipseşte) în declaraţia funcţiei (în declaraţie nu se dau parametrii formali)
56 listă eronată a parametrilor formali ai funcţiei
57 tipul trebuie să fie funcţie
60 memorie plină

61 variabilele **auto** pot fi inițializate doar prin atribuire sau cu funcția **move**
62 clasa de memorie nu permite inițializarea
63 tipul nu permite inițializarea
64 prea multe date de inițializare

5.6 Manipularea datelor în memorie

5.6.1 Dimensiunea tipurilor

char	1 byte
int	2 bytes, byte-ul semnificativ la adresa superioară (convenția Intel)
short	identic int
long	identic int
unsigned	identic int
pointer	2 bytes , byte-ul semnificativ la adresa superioară (conventia Intel)
tablouri	nr-elemente * nr-bytes-pe-element elementul inferior la adresa inferioară variabilele multiplu indexate sînt considerate tablouri de tablouri
struct	numărul de bytes necesari este egal cu suma numărului de bytes necesari fiecărui membru, primul membru la adresa inferioară
union	numărul de bytes este egal cu numărul de bytes ocupați de variabila cea mai extinsă, membrii sînt aliniați la adresa inferioară

5.6.2 Clase de memorie

constante	sînt incluse în programul compilat
extern,static	începînd de la adresa dată de variabila-sistem RAMTOP în jos, adresele sînt incluse în codul compilat
auto	sînt puse pe stivă și accesate prin regiștrii IX și SP

5.6.3 Funcții

Parametrii efectivi ai funcțiilor sînt evaluați de la stînga la dreapta și puși pe stivă în aceeași ordine. Pentru funcțiile cu număr variabil de argumente se trimit 2 bytes suplimentari care reprezintă numărul de bytes ai argumentelor și sînt utilizați pentru accesarea argumentelor și resetarea stivei.

Funcţiile sînt apelate prin instrucţiunea CALL, se încarcă registrul SP în IX apoi se alocă spaţiu în continuare pe stivă pentru variabilele proprii (locale).

5.6.4 Fişiere

Toate fişierele (inclusiv programul-sursă) sînt manipulate ca şiruri de caractere. Fisierele pe banda magnetica au un header de tip Spectrum şi un şir de blocuri de 512 bytes fiecare din care primii 2 bytes reprezintă un contor de caractere; bitul cel mai semnificativ al contorului ultimului bloc este setat.

5.6.5 Regiştrii Z80

HL este utilizat pentru returnarea rezultatului funcţiilor şi evaluarea expresiilor. A', F', B', C', D', E', H', L', IY, I şi R nu sînt utilizaţi de compilator sau de codul compilat.

6 FORTH

6.1 Instalarea*

Alături de Pascal, Forth este cel mai eficient limbaj de programare pe calculatoarele compatibile Spectrum, fiind însă mai dificil de utilizat.
După încărcarea cu **LOAD ""** prompt-ul C asteaptă introducerea comenzilor (mod interpretor).

6.2 Structuri de limbaj

Forth poate lucra în mod interpretor, cînd aşteaptă introducerea unor comenzi de la tastatură, sau mod compilator, cînd compilează programele.
Forth poate manipula 2 stive:

stiva de date	este cea curenta; notaţie SD
stiva de retur	notaţie SR
Notaţie pentru stive:	(înainte-de-execuţie — — după-execuţie)
	elementul din dreapta este în virful stivei

Programul constă în definirea succesivă a unor cuvinte (subrutine) pînă la definirea cuvîntului de nivel cel mai înalt care prin execuţie generează rezultatul programului.
Un cuvînt poate fi definit cu ajutorul cuvîntului de definiţie : :

```
: cuvînt-de-definit
      cuvînt-1
```

*Fig-Forth este produs înregistrat al firmei ABERSOFT

...
cuvînt-n
;

unde *cuvînt-1 ...cuvînt-n* sînt cuvinte definite anterior (în acelaşi vocabular sau în vocabulare ierarhic superioare), ; este tot un cuvînt.
Un cuvînt poate fi definit şi cu ajutorul cuvintelor de definiţie (predefinite în Forth sau definite de utilizator)

cuvînt-de-definiţie cuvînt-de-definit
cuvînt
...

Cuvintele sînt şiruri de caractere afişabile, separate de blank-uri, de orice lungime, dar din care doar primele 31 de caractere sînt semnificative.
Numerele sînt şiruri de cifre şi pot fi exprimate în orice bază de la 0 la 36; pentru baze superioare lui 10 se utilizează literele ca cifre.
Distincţia între cuvinte şi numere se face astfel:
a) la introducerea unui şir de caractere (de la consolă sau RAM-DISC) se încearcă interpreterea primului şir ca un cuvînt prin căutarea acestuia în dicţionar, începînd de la ultimul cuvînt, definit (în vocabularul de context);
b) dacă şirul nu este cuvînt din dicţionar se încearcă interpretarea acestuia ca un număr în baza curentă; dacă încercarea este reuşită numărul, transformat în corespondentul binar, este înpins pe vîrful stivei SD; numerele în dublă lungime (pe 4 bytes) sînt semnalizate prin introducerea caracterului . în orice poziţie
c) dacă şirul nu este nici număr atunci se generează mesaj de eroare
Expresiile se scriu în notaţie postfixată (operatorul la sfîrşit) astfel încît toţi operatorii au aceeaşi prioritate şi se evaluează de la stînga la dreapta (operatorii sînt tot cuvinte).

6.3 Structura memoriei

6.3.1 Harta memoriei

5E00	Variabile-utilizator
5E40	Interpretorul de adrese
	PUSHDE, PUSHL, NEXT
5E77	Dicţionar
	FENCE, DP
	HERE
	PAD
	Stiva-date
	S0, TIB
CB40	Buffer-terminal
	Stiva-retur
	R0

CBE0 Buffer-intrare/ieşire cu RAM-DISC
FIRST
LIMIT
D000 RAM-DISC
FC00 UDG

Între 5E00 şi 5E31 (inclusiv) sînt variabilele-utilizator predefinite (celelalte 10 locaţii sînt libere) care sînt utilizate la fel ca variabilele normale, dar sînt definite printr-un deplasament faţă de adresa de început a tabelei (5E00).
Forth simulează un RAM-DISC de 10 kbytes (10 ecrane) şi care poate fi salvat sau încărcat pe bandă magnetică.

6.3.2 Structura cuvintelor

1)Cuvinte definite prin :

NFA byte-def
byte-caracter-1
...
byte-caracter-n
LFA NFA-cuvînt-precedent
CFA CFA-:
PFA CFA-cuvînt
...
CFA-;S

unde byte-def are structura:

7	6	5	4	3	2	1	0
1	P	S		lungime-nume (n)			

P = bit de precedenţă, P = 1 pentru cuvinte imediate
S = bit de validare a definiţiei, S = 0 pentru definiţie validă

2)Variabile

Antetul unui cuvînt-variabilă de la NFA la CFA este identic; urmează 2 bytes adresa de tratare a variabilei, apoi 2/4 bytes pentru memorarea valorii curente a variabilei.

3)Constante

Antetul unui cuvînt-constantă de la NFA la CFA este identic; urmează 2 bytes adresa de trateare a constantei, apoi 2/4 bytes pentru memorarea valorii constantei.

4)Vocabulare

Antetul unui cuvînt-constantă de la NFA la CFA este identic; urmează:

CFA adresa de tratare pentru cuvinte de definiţie (DOES>)
PFA adresa de tratare pentru cuvinte-vocabular
PFA+2 A081, antet de cuvînt fictiv
PFA+4 NFA al ultimei definiţii făcută sub vocabularul respectiv
PFA=6 legatura către vocabularul precedent în care a fost creeat (FORTH fiind primul vocabular conţine 0)

5)Cuvinte scrise în cod-maşină

Antetul unui cuvînt scris în cod maşină de la NFA la CFA este identic; în CFA este adresa CFA+2, urmează o secvenţă în cod-maşină care se termină cu un salt la adresa NEXT, PUSHHL sau PUSHDE.

6.4 Editorul

Editorul este scris în Forth (cuvintele editorului sînt definite în Forth), este construit ca vocabular separat cu numele **EDITOR**, şi accesul în el se face prin tastarea **EDITOR** (urmat de Enter).
Programul-sursă este organizat în ecrane avînd numerele între 0 şi 32767. Fiecare ecran are 16 linii numerotate de la 0 la 15 cu pînă la 64 de caractere pe fiecare linie.

Notaţie:
- n — număr cu semn pe 2 bytes
- adr — adresa pe 2 bytes
- c — caracter, cod ASCII, 1byte
- text — şir de caractere afişabile

#LAG (— —adr n)
adr adresa cursorului
n numărul caracterelor rămase pînă la sfîrşit de linie

#LEAD (— —adr n)
adr adresa liniei
n deplasamentul cursorului în linie

#LOCATE (— —n1 n2)
n1 numărul de linie unde este cursorul
n2 deplasamentul cursorului în linie

B (— —)
mişcă cursorul înapoi cu lungimea textului din PAD

C *text* (— —)
inserează *text* după poziţia cursorului

CLEAR (n— —)
şterge ecranul n

COPY (n1 n2— —)
copiază ecranul n1 în ecranul n2

D (n— —)
şterge linia n dar o reţine în PAD

DELETE (n— —)
şterge n caractere de la poziţia cursorului

E (n— —)
şterge linia n cu blank-uri

F *text* (— —)
introduce text în PAD şi îl caută în ecranul curent

FIND (— —)
caută textul din PAD în ecranul curent de la poziţia cursorului

H (n— —)
copiază conţinutul liniei n din ecranul curent în PAD

I (n— —)

introduce textul din PAD în linia n din ecranul curent

L (— —)

listează ecranul curent

1LINE (— —n)

caută în linia curentă de la poziţia curentă a cursorului textul din PAD şi returnează n=1 dacă îl găseşte, altfel n=0

M (n— —)

avansează cursorul cu n caractere

MATCH (adr1 n1 adr2 n2— —n3 n4)

caută textul care începe la adr1 de lungime n1 în textul care începe la adr2 pe lungime n2; revine cu n4=1 dacă textul a fost găsit şi n3 deplasamentul de unde începe textul

-MOVE (adr n— —)

copiază o linie de text care începe la adr în linia n a ecranului curent

N (— —)

caută în continuare textul din PAD

P *text* (n— —)

pune *text* în linia n

R (n— —)

înlocuieşte linia n cu textul din PAD

S (n— —)

inserează o linie vidă n,următoarele linii se mută în jos, ultima linie se pierde

T (n— —)

afişează linia n din ecranul curent şi o introduce în PAD

-TEXT (adr1 n1 adr2— —n2)

compară 2 texte de la adr1 şi adr2 pe lungimea n1, revine cu n2=1 dacă sînt egale, altfel n2=0

TILL *text* (— —)

şterge caracterele de la poziţia cursorului la *text*

TOP (— —)

poziţionează cursorul la începutul ecranului curent

X *text* (— —)

caută un *text* de la cursorul curent şi sterge *textul*

6.5 Cuvinte predefinite

Notaţii:

Pentru date:

adr	adresă de memorie, 2 bytes
b	byte
c	cod de caracter ASCII, 1 byte
d	număr cu semn în dublă lungime, 4 bytes
f	fanion logic, f=0 fals, f=1 adevărat, 2 bytes
n	număr cu semn, 2 bytes
u	număr fără semn, 2 bytes

ud număr fără semn în dublă lungime, 4 bytes
text şir de caractere afişabile
listă listă de cuvinte şi/sau numere, separate de blank-uri

Pentru cuvinte:

C poate fi utilizat numai în mod compilare (definiţii)
E poate fi utilizat numai în mod execuţie (interpretor)
I este imediat (are bitul de precedenţă setat)
U variabilă-utilizator, memorarea lor este diferită de cele normale dar sînt utilizate identic
F scris în Forth
m scris în cod-maşină

! (n adr— —) m
memorează n la adr

!CSP (— —) F
memorează pointerul SD în variabila CSP

(ud1— —ud2) F
generează din ud1 o cifră caracter ASCII în şirul de ieşire; se utilizează pentru formatarea numerelor, numai între <# şi #>

#> (ud— —adr n) F
termină formatarea numerelor, revine cu adresa şi lungimea şirului, paramertii utilizabili de TYPE

#BUFF (— —n)
constantă (utilizată de FLUSH)

#S (ud— —00) F
converteşte toate cifrele din ud în şir de caractere ASCII, se utilizează numai între <# şi #>

' *text* (— —PFA) F
returnează PFA al cuvîntului *text*

((— —) F, I , C
început de comentariu care se termină cu caracterul)

(.") (— —) F
cuvînt compilat în definiţii de către ."

(;CODE) (— —) F
cuvînt compilat în definiţii de către ;CODE

(+LOOP) (— —) m,C
cuvînt compilat în definiţii de către +LOOP

(ABORT) (— —) F
identic cu **ABORT**

(DO) (n1 n2— —) m
cuvînt compilat în definiţii de către **DO**

(FIND) (adr NFA— —PFA b 1) m
sau (adr NFA— —0)
caută în dicţionar începînd de la NFA şirul care începe la adr, revine cu PFA, lungimea şirului şi fanion adevărat dacă l-a găsit, altfel fanion fals

(LINE) (n1 n2— —adr n) F,C
n1 număr de linie, n2 număr de ecran
adr adresă şirului de caractere, n lungimea şirului de caractere al liniei

(LOOP) (— —) m,C
cuvînt compilat în definiţii de către LOOP

(NUMBER) (d1 adr— —d2 adr2) F
converteşte textul de la adr1 + 1, conform BASE, într-un număr acumulat în

d1 şi lăsat ca d2; adr2 este adresa primului caracter neconvertibil

(TAPE) (n— —) m

rutină folosită de LOADT, SAVET şi VERIFY pentru manipularea RAM-DISC-ului pe bandă magnetică:
n=0 se realizează salvare (SAVET)
n=1 se realizează încărcare (LOADT)
n=2 se realizează verificare (VERIFY)
RAM-DISC-ul este salvat pe bandă magnetică într-un fişier cu numele DISC; header-ul este construit la adresa 75E6 (un alt header se află la adresa 75F7)

***** (n1 n2— —n3) F

n3=n1*n2

***/** (n1 n2 n3— —n4) F

d=n1*n2; n4=d/n3

***/MOD** (n1 n2 n3— —n4 n5) F

d=n1*n2; n4=d/n3; n5=restul impartirii d/n3

+ (n1 n2— —n3) F

n3=n1+n2

+! (n adr— —) m

adaugă n la valoarea conţinută la adr

+- (n1 n2— —n3) F

aplică semnul lui n2 asupra lui n1 care devine n3

+BUF (adr— —f) F

cuvînt utilizat de BUFFER

+LOOP (n— —) F

marchează sfîrşitul buclei cu contor: **DO** *cuvînt* ... **+LOOP**
incrementează contorul cu n şi iese din buclă dacă:
n>0 şi contor >=valoarea-limită sau
n<0 şi contor <= valoarea-limită

+ORIGIN (n— —adr) F

adaugă n la valoarea 5E40 (adresa de început a interpretorului de adrese)

, (n— —) F

introduce n în următoarea intrare liberă din dicţionar

- (n1 n2— —n3) F

n3=n1-n2

---> (— —) F

continuă interpretarea din ecranul următor

-DUP (n— —n n) F
sau (0— —0)

duplică elementul din virful stivei dacă este nenul

-FIND *text* (— —1 n PFA) F,C
sau (— —0)

caută cuvîntul *text* în dicţionar, revine cu lungimea şi PFA, altfel f=0

-TRAILING (adr1 n1— —adr2 n2) F

elimină blank-urile din şirul de la adr1 de lungime n1 şi revine cu adresa şi lungimea şirului format

. (n— —) F

afişează n conform BASE

." *text* (— —) F, I

afişează textul care se termină cu caracterul "

.CPU (— —) F

afişează mesajul 48k SPECTRUM

.LINE (n1 n2— —) F
afişează linia n1 din ecranul curent

.R (n1 n2— —) F
afişează numărul n1, conform BASE pe n2 spaţii, aliniat la dreapta

/ (n1 n2— —n3) F
n3=n1/n2

/MOD (n1 n2— —n3 n4) F
n4=n1/n2, n3=restul împărţirii n1/n2

0 (— —0)
constantă

1 (— —1)
constantă

2 (— —2)
constantă

3 (— —3)
constantă

0< (n— —f) m
f=1 dacă n<0, altfel f=0

0= (n— —f) m
f=1 dacă n=0, altfel f=0

0BRANCH (f— —) m,C
salt relativ conform numărului, cu semn, în complement faţă de 2, care urmează în definiţia compilată a unui cuvînt dacă f<>0; cuvînt compilat, în definiţii, de IF, UNTIL, WHILE

1+ (n1— —n2) F
n2=n1+1

2! (d adr— —) m
memorează d la adr

2@ (adr— —d) m
citeşte d de la adr

2+ (n1— —n2) F
n2=n1+2

2CONSTANT *text* (d— —) F/m
creează o constantă în dublă lungime, căreia i se atribuie valoarea d; execuţia ulterioară a cuvîntului *text* determină plasarea d pe stiva SD

2DROP (d— —) F
şterge d de pe SD

2OVER (d1 d2— —d1 d2 d3) F
duplică penultimul element pe SD

2SWAP (d1 d2— —d2 d1) F
interschimbă ultimele 2 elemente din vîrful stivei

2VARIABLE *text* (d— —) F/m
creează o variabilă în dublă lungime căreia i se atribuie ca primă valoare d; execuţia cuvîntului *text* determină plasarea adresei variabilei pe stiva SD

: (— —) F/m, I, E
: *cuvînt-de-definit cuvînt ...* **;**
semnalizează începutul definiţiei unui cuvînt; Forth trece în mod compilare; vocabularul CONTEXT devine CURRENT

; (— —) F, I, C
indică sfîrşitul definiţiei (:) unui cuvînt, Forth trece în mod interpretor, compilează **;S** la sfîrşitul cuvîntului

;CODE (— —) F, C, I

indică faptul că în definiția unui cuvînt urmează o secvență în cod-mașină. Forth utilizează regiștrii Z80 astfel:

A registru de uz general
BC conține IP, pointer către adresa CFA a următorului cuvînt de executat
DE registru de uz general
HL registru de uz general, prin intermediul acestuia se manipulează SR al cărui pointer se află la adresa 5E68.
IX conține 5E00, pointer către tabela variabilelor-utilizator
IY nu este folosit
SP manipulează SD

setul alternat de regiștri nu este folosit; secvența de cod-mașină trebuie să se termine cu un salt la una din adresele date de NEXT, PUSHHL, PUSHDE

;S (— —) m

cuvînt folosit pentru terminarea compilării programului-sursă din ecrane, redînd controlul consolei; este compilat de : la sfîrșitul definițiilor

< (n1 n2— —f) m

f=1 dacă n1 < n2 altfel f=0

<# (d— —d) F

semnalizează începutul formatării numerelor

<BUILDS (— —) F, C

utilizat pentru definirea cuvintelor de definiție împreună cu DOES>; semnalizează începutul secvenței executate la compilare de către cuvîntul de definiție

```
: cuvînt-de-definiție
      <BUILDS listă1
      DOES> listă2 ;
```

= (n1 n2— —f) F

f=1 dacă n1=n2, altfel f=0

> (n1 n2— —f) F

f=1 dacă n1>n2, altfel f=0

>R SD (— —n) și SR (n— —) m, C

transferă vîrful SR pe SD, se utilizează în pereche cu **R >**

? (adr— —) F

afișează ca număr cu semn valoarea aflată la adr

?COMP (— —) F

testează dacă Forth se află în mod compilare, afișează mesaj de eroare dacă nu

?CSP (— —) F

afișează mesaj de eroare dacă pointerul stivei diferă de CSP

?ERROR (f n— —) F

afișează mesaj de eroare n dacă f=1

?EXEC (— —) F

afișează mesaj de eroare dacă Forth nu este în mod interpretor (execuție)

?LOADING (— —) F

testează dacă blocul de disc este încărcat în memorie, dacă BLK=0 emite mesaj de eroare

?PAIRS (n1 n2— —) F

emite mesaj de eroare dacă n1 < >n2 (utilizat pentru verificări sintactice)

?STACK (— —) F

verifică dacă stiva se află în limitele permise, emite mesaj de eroare în caz contrar

?TERMINAL (— —f) F

f=1 dacă a fost apăsată tasta Break

@ (adr — —n) m

citeşte n de la adr

ABORT (— —) F

resetează SD şi SR, comută Forth în mod interpretor, repune BASE pe DECIMAL, vocabularul CONTEXT şi CURRENT devine FORTH

ABS (n1— —n2) F

returnează n2 modulul numărului n1

AGAIN (— —) F

marchează sfîrşitul buclei infinite: **BEGIN** *listă* **AGAIN**

ALLOT (n— —) F

rezervă n bytes în dictionar, începînd cu prima intrare liberă

AND (n1 n2— —n3) m

efectuează şi-logic pe biţi între n1 şi n2 returnînd rezultatul n3

AT (n1 n2— —) F

poziţionează cursorul de afişare pe linia n1 (între 0 şi 31) şi coloana n2 (între 0 şi 31)

ATTR (n1 n2— —b) F

returnează atributul de culoare al unui caracter aflat în rîndul n1 (între 0 şi 21) şi coloana n2 (între 0 şi 31)

biţi 0-2 codul de culoare INK
biţi 3-5 codul de culoare PAPER
bit 6 cod BRIGHT , 0 fără , 1 cu
bit 7 cod FLASH, 0 fără, 1 cu

cod de culoare

0 negru	4 verde
1 albastru	5 cyan
2 roşu	6 galben
3 magenta	7 alb

B/BUF (— —n)

n=128 lungimea buffer-ului, în bytes, utilizat în operaţii cu memoria externă (RAM-DISC); constantă

B/SCR (— —n)

numărul de buffere de ecran; constantă

BACK (adr— —) F

calculează saltul relativ între HERE şi adr şi îl compilează (după BRANCH sau 0BRANCH)

BASE (adr— —) U

variabilă-utilizator care conţine baza curentă de numeraţie

BEGIN (— —adr) F, C, I

marchează începutul unei bucle:

BEGIN *listă* **AGAIN**
BEGIN *listă* **UNTIL**
BEGIN *listă1* **WHILE** *listă2* **REPEAT**

la compilare lasă pe stivă adresa HERE utilizată (la compilare) de cuvintele de sfîrşit de buclă

BL (— —n)

n=32, codul ASCII pentru blank; constantă

BLANKS (adr n— —) F

depune în memorie începînd de la adr, pe n bytes, codul ASCII al blank-ului

BLEEP (n1 n2— —) m

emite un sunet la difuzor
n1 = f*t unde f este frecventa sunetului (Hz) şi t durata lui (s)
n2 = 3.5*106/8/f-30.125

do	261.63Hz	fa#	369.99Hz
do#	277.18Hz	sol	392Hz
re	293.66Hz	sol#	415.30Hz
re#	311.13Hz	la	444Hz
mi	329.63Hz	la#	466.16Hz
fa	349.23Hz	si	493.88Hz

la fiecare salt la octava superioară/inferioară frecventele se dublează/înjumătăţesc

BLK (— —adr) U

variabilă-utilizator care conţine numărul de bloc care se interpretează; dacă este 0 interpretarea se face de la consolă

BLOCK (n— —adr) F

returnează adresa blocului n

BORDER (n— —) m

setează culoarea BORDER conform n:

0 negru	4 verde
1 albastru	5 cyan
2 rosu	6 galben
3 magenta	7 alb

BRANCH (— —) m, C

salt relativ conform numărului (cu semn, în complement faţă de 2) care urmează în definiţia compilată a unui cuvint; cuvînt compilat (în definiţii) de AGAIN, ELSE, REPEAT

BRIGHT (n— —) F

strălucirea următoarelor caractere afişate este accentuată dacă n = 1 sau revine la normal dacă n = 0

BUFFER (n— —adr) F

revine cu adresa următorului buffer disponibil şi îl asignează numărului de bloc n

C! (b adr— —) m

memorează b la adr

C, (b— —) F

memorează b în prima locaţie liberă din dicţionar

C@ (adr— —b) m

citeşte b de la adr

C/L (— —n)

n = 64, numărul de caractere pe linie; constantă

CASE (n— —) F, C, I

marcator de început al unei ramificaţii:

CASE *n1* **OF** *listă1* **ENDOF**
...
n2 **OF** *listă2* **ENDOF**
listă **ENDCASE**

dacă n corespunde cu una din valorile *n1*, ... *nk* se execută secvenţa corespunzătoare, dacă nu, se execută secvenţa de dinainte de ENDCASE care este opţională; în toate cazurile execuţia continuă cu cuvintele de

după **ENDCASE**

CFA (PFA— —CFA) F

returnează CFA al cuvîntului al cărui PFA se află pe SD

CLS (— —) m

şterge ecranul

CMOVE (adr1 adr2 n— —) m

mută n bytes de la adr1 la adr2, transferul se execută începînd cu byte-ul de la adr1

COLD (— —) F

iniţializare totală a sistemului Forth

COMPILE *text* (— —) F, C

compilează CFA al cuvîntului *text* în următoarea intrare liberă din dicţionar

CONSTANT *text* (n— —) F/m

creează o constantă căreia i se atribuie valoarea n; execuţia ulterioară a *text* determină plasarea n pe SD

CONTEXT (—.—adr) U

variabilă-utilizator care conţine NFA al ultimului cuvînt definit în vocabularul CONTEXT, în care are loc căutarea cuvintelor în mod interpretor

COUNT (adr— —adr+1 n) F

la adr se află un şir de caractere, în primul byte se află lungimea şirului, în următorii,codurile ASCII; COUNT revine cu parametrii utilizabili de TYPE

CR (— —) m

efectuează un retur de car pe ecran

CREATE *text* (— —) F

creează o intrare în dictionar care conţine cîmpurile NFA, LFA, CFA; în CFA este introdusă adresa CFA+2 adică PFA; cîmpurile PFA nu sînt completate

CSP (— —adr) U

variabilă-utilizator care conţine pointerul SD

CURRENT (— —adr) U

variabilă-utilizator care conţine NFA al ultimului cuvînt definit în vocabularul CURRENT la care se adaugă definiţiile compilate

D+ (d1 d2— —d3) m

d3=d1+d2

D+- (d1 d2— —d3) F

aplică semnul lui d2 asupra lui d1 care devine d3

D. (d— —) F

afişează d conform BASE

D.R (d n— —) F

afişează numărul d conform BASE pe n spaţii, aliniat la dreapta

DABS (d1— —d2) F

returnează d2 modulul numărului d1

DECIMAL (— —) F

setează BASE la 10

DEFINITIONS (— —) F

declară vocabularul CONTEXT ca vocabular CURRENT

DIGIT (c n1— —n2 1) m

sau (c n1— —0)

converteşte c în n2 conform bazei n1şi lasă un fanion pentru semnalizarea validării operaţiei de conversie

DLITERAL (d— —) mod compilare F, I

(— —d) mod execuţie

compilează ca literali numerele care apar în definiții; la execuția definițiilor numerele sînt plasate pe SD

DMINUS (d1— —d2) m

returnează d2 complement față de 2 a lui d1 (înmulțire cu -1)

DO SD (n1 n2— —) m
SR (— —n2 n1)

marcator de început al buclei cu contor:
DO *listă* **LOOP** sau **DO** *listă* **+LOOP**
n1 reprezintă valoarea inițiala a contorului
n2 reprezintă valoarea finală a contorului

DOES > (— —) F/m

marcator de început al secvenței realizate la execuție în definiția cuvintelor de definiție:
: *cuvînt-de-definiție listă1* **DOES >** *listă2* **;**

DP (— —adr) U

variabilă-utilizator care conține pointerul curent al dicționarului

DPL (— —adr) U

variabilă-utilizator care conține deplasamentul punctului la introducerea numerelor; DPL> =0 pentru număr cu punct, DPL<0 pentru număr fără punct, valoare implicită: -1

DR0 (— —) F

resetează la 0 variabila-utilizator OFFSET

DRAW (n1 n2— —) F

trasează o dreaptă din ultima poziție PLOT sau DRAW anterioară, implicit colțul stînga jos al ecranului pînă în punctul de coordonate absolute n1 (de la stînga la dreapta, între 0 și 255) și n2 (de jos în sus, între 0 și 183); depășirile marginilor sînt ignorate; CLS resetează poziția la starea implicită

DROP (n— —) m

șterge n de pe SD

DUP (n— —n n) m

duplică n pe SD

EDITOR (— —)

vocabularul EDITOR devine vocabular CONTEXT

ELSE (— —) F, C, I

marchează începutul ramurii de condiție falsă, opțională, din structura IF

EMIT (c— —) m

afișează caracterul c pe ecranla poziția curentă a cursorului

EMPTY-BUFFERS (—) F

inițializează zona buffer-elor de disc dintre FIRST și LIMIT

ENCLOSE (adr1 c— —adr2 n1 n2 n3) m

cuvînt utilizat de WORD pentru interpretarea unui șir de cuvinte de la adr, cu delimitatorul c

END (— —) F, C, I

identic cu UNTIL

ENDCASE (— —) F, C, I

marchează sfîrșitul structurii CASE

ENDIF (— —) F, C, I

identic cu THEN

ENDOF (— —) F, C, I

marchează sfîrșitul unei ramuri de execuție în structura CASE

ERASE (adr n— —) F

șterge cu 0, n bytes începînd de la adr

ERROR (n1— —n2 n3) F
cuvînt utilizat de ?ERROR

EXECUTE (CFA— —) m
execută cuvîntul al cărui CFA se află pe SD

EXPECT (adr n— —) F
aşteaptă introducerea a n caractere de la tastatură, ale căror coduri ASCII le depune în memorie începînd de la adr; şirul poate fi mai scurt dacă se termină cu Enter

EXIT (— —) F
forţează terminarea prematură a cuvîntului în care se află

FENCE (— —adr) U
variabilă-utilizator care conţine adresa limită pînă la care se pot şterge cu FORGET definiţiile din dictionar

FILL (adr n b— —) m
iniţializează cu b, n bytes începînd de la adr

FIRST (— —adr)
constantă, conţine adresa primului buffer de disc

FLASH (n— —) F
următoarele caractere afişate sînt clipitoare dacă n=1; cu n=0 se revine la normal

FLD (— —adr) U
variabilă-utilizator care conţine pointerul zonei de conversie a numerelor date la afişare

FLUSH (— —) F
rescrie toate buffer-ele de disc care au setate indicatorul de punere la zi cu UPDATE anterior

FORGET *text* (— —) F
şterge toate definiţiile ulterioare, inclusiv *text*, din dictionar; vocabularul CURRENT trebuie să fie acelaşi cu CONTEXT

FORTH (— —)
reprezintă numele vocabularului de bază, în care se intră la iniţializarea sistemului ca vocabular CONTEXT şi CURRENT

FREE (— —n) F
returnează numărul n de bytes liberi din dictionar

GOVER (n— —) F
n=1 permite suprascrierea în ecran peste caracterele afişate anterior, n=0 revine la normal

HERE (— —adr) F
returnează adresa primei celule libere din dictionar

HEX (— —) F
setează BASE la 16

HLD (— —adr) U
variabilă-utilizator care conţine pointerul ultimului caracter introdus în PAD

HOLD (c— —) F
inserează caracterul c în şirul unui număr formatat, se utilizează numai între <# şi #>

HI (— —n)
constantă, n=FBFF, ultima adresă a RAM-DISC-ului

I (— —n) m, C
copiază pe SD vîrful stivei SR

I' (— —n) m, C
copiază pe SD penultimul element de pe SR

ID. (NFA— —) F
afişează numele cuvîntului al cărui NFA se află pe SD

IF (f— —) F, C, I
marchează începutul ramurii de condiţie adevărata din structurile: **IF** *listă* **THEN** sau **IF** *listă1* **ELSE** *listă2* **THEN**; în ambele cazuri execuţia continuă cu cuvintele aflate după THEN

IMMEDIATE (— —) F
marchează drept imediat ultimul cuvînt definit; cuvintele imediate sînt executate în decursul compilării dacă execuţia lor nu a fost inhibată cu [COMPILE]

IN (— —adr) U
variabilă-utilizator, conţine un offset care este o adresă relativă a următorului text de interpretat de către WORD

INCX (— —adr)
variabilă în dublă lungime utilizată de DRAW

INCY (— —adr)
variabilă în dublă lungime utilizată de DRAW

INDEX (n1 n2— —) F
afişează liniile 0 ale ecranelor cu numerele între n1 şi n2

INIT-DISC (— —) F
iniţializează (şterge) RAM-DISC-ul

INK (n— —) F
setează culoarea INK conform n:

0 negru	4 verde
1 albastru	5 cyan
2 roşu	6 galben
3 magenta	7 alb

INTERPRET (— —) F
lansează interpretarea şirului de intrare din blocul BLK (BLK=0: interpretarea se face din TIB)

INKEY (— —c) m
returnează codul ASCII al tastei apăsate (fără a aştepta apăsarea tastei); returnează 255 dacă nu a fost apăsata nici o tastă

INP (u1— —u2) m
citeşte portul u1 şi returnează valoarea u2 citită; se realizează instrucţiunile: **LD BC,** *u1* ; **IN A, (C)**; conţinutul A este depus pe SD

INVERSE (n— —) F
următoarele caractere afişate sînt INVERSE dacă n=1; cu n=0 se revine la normal

J (— —n) F, C
copiază antepenultima valoare de pe SR pe SD

KEY (— —c) m
aşteaptă apăsarea unei taste şi returnează codul ASCII al tastei apăsate

LATEST (— —NFA) F
returnează NFA al ultimei definiţii din vocabularul CURRENT

LEAVE (— —) m, C
forţează terminarea buclelor cu contor: DO ... LOOP/+LOOP la următoarea execuţie a cuvîntului LOOP/+LOOP (prin egalarea valorii curente a contorului cu valoarea finală)

LFA (PFA— —LFA) F
returnează adresa LFA a cuvîntului a cărui adresă PFA este pe SD

LINE (n— —) F

returnează adresa liniei n din ecranul curent

LINK (n— —) m

n=1 cuplează imprimanta, n=0 decuplează imprimanta

LIMIT (— —n)

constantă, conţine adresa ultimei locaţii de memorie +1

LIST (n— —) F

listează ecranul n care devine ecran curent

LIT (— —) m, C

cuvînt compilat de LITERAL, în faţa unui număr, în dicţionar; în mod execuţie LIT plasează pe SD numărul care urmează (după LIT) în definiţia compilată

LITERAL (n— —) mod compilare F, C
(— —n) mod execuţie

compilează numerele care apar în definiţii ca literali; la execuţie aceste numere sînt plasate pe SD

LO (— —n)

constantă, n=D000 prima adresă a RAM-DISC-ului

LOAD (n— —) F

lansează interpretarea din ecranul n, iniţializează BLK, IN

LOADT (— —) F

încarcă de pe bandă magnetică un fişier DISC în RAM-DISC

LOOP (— —) F

marchează sfîrşitul buclei cu contor: **DO** *listă* **LOOP**; incrementează contorul buclei cu 1 şi iese din buclă dacă contorul curent este egal sau mai mare ca valoarea limită

M* (n1 n2— —d) F

d=n1*n2

M/ (d n1— —n2 n3) F

n3=d/n1, n2=restul împărţirii d/n1

M/MOD (ud1 u1— —u2 ud2) F

ud2=ud1/u1, u2=restul împărţirii ud1/u1

MAX (n1 n2— —n3) F

n3 maxim dintre n1 şi n2

MESSAGE (n— —) F

utilizat de ?ERROR, afişează locul erorii şi mesajul MSG# n, n fiind codul erorii

MIN (n1 n2— —n3) F

n3=minim dintre n1 şi n2

MINUS (n1— —n2) m

returnează n2 complement faţă de 2 a lui n1 (înmulţire cu -1)

MOD (n1 n2— —n3) F

n3=restul împărţirii n1/n2

MON (— —) m

reîntoarcere în BASIC Spectrum (cu mesajul de eroare **9: STOP statement 2:1**); relansarea Forth se face **GO TO 2** (COLD) sau **GO TO 3** (WARM)

NEXT (— —n)

constantă, n=5E6C adresa de reîntoarcere dintr-o secvenţă în cod-maşină (adresa interpretorului de adrese)

NFA (PFA— —NFA) F

revine cu adresa NFA a cuvîntului a cărui adresa PFA se află pe SD

NOOP (— —) F

instrucţiune vidă

NOT (f1— —f2) F
inversează fanionul logic f1, returnînd f2; identic cu 0=

NUMBER (adr— —d) F
convertește un șir de caractere, terminate cu blank, de la adr, în număr pe dublă lungime conform BASE; emite un mesaj de eroare dacă operația nu este posibilă

OF (— —) F, C, I
cuvînt utilizat în structura CASE, marchează începutul unei ramuri de prelucrare

OFFSET (— —adr) U
variabilă-utilizator care conține un offset pentru calculul adresei fizice a unui bloc

OR (n1 n2— —n3) m
efectuează sau-logic pe biți între n1 și n2 returnînd rezultatul n3

OUT (— —adr) U
variabilă-utilizator utilizată de EMIT

OUTP (u1 u2— —) m
trimite la portul u2 byte-ul u1; se realizează instrucțiunile: **LD BC,** *u2*; **LD A,** *u1*; **OUT (C), A**

OVER (n1 n2— —n1 n2 n1) m
copiază în vîrful SD penultimul element

PAD (— —adr) F
returnează adresa primei locații PAD (HERE+70)

PAPER (n— —) F
setează culoarea PAPER conform n:

0 negru	4 verde
1 albastru	5 cyan
2 roșu	6 galben
3 magenta	7 alb

PLOT (n1 n2— —) m
afișează un punct de coordonate:
n1, între 0 și 255, de la stînga la dreapta
n2, între 0 și 183, de jos în sus
depășirile marginilor sînt ignorate

PFA (NFA— —PFA) F
revine cu adresa PFA a cuvîntului a cărui adresa NFA se află pe SD

POINT (n1 n2— —f) m
returnează f=1 pentru pixel aprins (de culoare INK) unde n1, n2 coordonatele pixel-ului:
n1, între 0 și 255, de la stînga la dreapta
n2, între 0 și 183, de jos în sus
depășirile marginilor sînt ignorate

PREV (— —adr) U
variabilă-utilizator utilizată de +BUF și UPDATE

PUSHDE (— —n)
constantă, n=5E6A, adresa de reîntoarcere dintr-o secvență în cod-masină (adresa interpretorului de adrese); regiștrii DE și HL sînt salvați pe SD (HL pe vîrful stivei SD)

PUSHHL (— —n)
constantă, n=5E6B, adresa de reîntoarcere dintr-o secvență în cod-mașină (adresa interpretorului de adrese); registrul HL este salvat pe SD

QUERY (— —) F

aşteaptă maxim 80 de caractere şi le depune în TIB, resetează IN

QUIT (— —) F

bucla sistemului de operare, iniţializează SD şi SR şi trece în mod execuţie (interpretor)

R (— —n) F, C

identic cu I

R# (— —adr) U

variabilă-utilizator utilizată de cuvintele din EDITOR

R/W (adr n f— —) F

procedură de scriere/citire pe RAM-DISC:
adr adresa buffer-ului de bloc
n numărul blocului
f=0/1 pentru scris/citit

R > SD (— —n) m, C
SR (n— —)

transferă vîrful SR pe SD; se utilizează în pereche cu R

R0 (— —adr) U

variabilă-utilizator care conţine adresa bazei SR

REPEAT (— —) F, C, I

marcator de sfîrşit al buclei: **BEGIN** *listă1* **WHILE** *listă2* **REPEAT**

ROT (n1 n2 n3— —n2 n3 n1) m

permută circular ultimele 3 elemente din SD

RPI (— —) m

iniţializează pointerul SR cu R0 (resetează SR)

RP@ (— —adr) m

returnează pointerul curent al SR

S->D (n— —d) m

transformă un număr în echivalentul său în dublă lungime

S0 (— —adr) U

variabilă-utilizator care conţine adresa bazei SD

SAVET (— —) F

salvează pe bandă magnetică RAM-DISC-ul într-un fişier cu numele DISC

SCR (— —adr) U

variabilă-utilizator care conţine numărul ecranului curent, actualizată de LIST

SCREEN (n1 n2— —c) m

returnează codul ASCII al caracterului afişat în poziţia:
n1, linia, între 0 şi 21, de sus în jos
n2, coloană, între 0 şi 31, de la stînga la dreapta

SIGN (n d1— —d2) F

inserează semnul -, dacă este cazul, în şirul unui număr formatat, se utilizează numai între <# şi #>

SIZE (— —n) F

returnează dimensiunea dicţionarului (împreună cu interpretorul de adrese)

SDMUDGE (— —) F

complementează bitul de validare al ultimei definiţii create

SPI (— —) m

iniţializează pointerul SD cu S0 (resetează SD)

SP@ (— —adr) m

returnează pointerul curent al SD (de dinaintea execuţiei cuvîntului)

SPACE (— —) F

afişează un blank pe ecran (în poziţia curentă)

SPACES (n— —) F

afişează n blank-uri pe ecran (din poziţia curentă)

STATE (— —adr) U

variabilă-utilizator care conţine starea Forth:
n=0/C0 mod interpretor/compilator

SWAP (n1 n2 — —n2 n1) m

permută ultimele 2 valori din virful SD

TEXT (c— —) F

transferă un text,delimitat de c, din TIB în PAD

THEN (— —) F, C, I

marcator de sfîrşit al structurilor IF ... THEN sau IF ... ELSE ... THEN

TIB (— —adr) U

variabilă-utilizator, conţine adresa buffer-ului de intrare de la tastatură

TOGGLE (adr n— —) m

realizează sau-exclusiv între byte-ul de la adr şi byte-ul mai puţin semnificativ din n, rezultatul este depus la adr

TRAVERSE (adr1 n— —adr2) F

caută capătul opus al numelui unui cuvînt Forth din dicţionar:
n=1/-1 căutarea se face către adrese superioare/inferioare
adr1 adresa unui capăt (NFA sau ultima litera din nume)
adr2 adresa celuilalt capăt

TRIAD (n— —) F

listează 3 ecrane: k, k+1, k+2 unde k este un număr divizibil cu 3 şi n este egal cu k, k+1 sau k+2

TYPE (adr n— —) F

afişează un şir de n caractere aflat în memorie la adr

U* (u1 u2— —u3) m

u3=u1*u2

U< (u1 u2— —f) F

f=1 dacă u1<u2, altfel f=0

U/MOD (ud u1— —u2 u3) m

u2=ud/u1, u3=restul împărţirii ud/u1

U. (u— —) F

afişează u conform BASE

U.R (u n— —) F

afişează u conform BASE pe n spaţii

UDG (— —adr) F

returnează adr zonei UDG

UNTIL (f— —) F, C, I

marcator de sfîrşit de buclă: **BEGIN** *listă* **UNTIL**; reia bucla de la BEGIN dacă f <>0 altfel continuă execuţia după UNTIL

UPDATE (— —) F

setează indicatorul de punere la zi al buffer-elor de disc care au fost modificate

USE (— —adr)

variabilă, adresa primului buffer de disc disponibil

USER *text* (n— —) F/m

creează o variabilă-utilizator cu numele *text*, n este adresa variabilei, offset faţă de adresa de început a tabelei (5E00); execuţia *text* plasează pe SD adresa (absolută) a variabilei

VARIABLE *text* (n— —) F/m

creează o variabilă cu numele *text* căreia îi atribuie valoarea inițială n;
execuția *text* plasează pe SD adresa (PFA) a variabilei

VERIFY (— —) F
verifică corectitudinea salvării pe bandă magnetică a fișierului DISC care conține RAM-DISC-ul

VOC-LINK (— —adr) U
variabilă-utilizator care conține adresa PFA+6 a ultimului vocabular creeat (utilizată pentru înlănțuirea vocabularelor)

VOCABULARY *text* (—) F
creează un nou vocabular cu numele *text*, vocabularele trebuie declarate ca fiind de tip imediat; vocabularul nou este creat în vocabularul CURRENT; execuția *text* determină ca acesta să devină vocabular CONTEXT; există 2 vocabulare predefinite: FORTH (de bază) și EDITOR

VLIST (—) F
listează toate cuvintele definite (începînd cu ultimul) din vocabularul CONTEXT

WARM (—) F
inițializarea sistemului Forth: șterge buffer-ele de disc, resetează SD și SR; vocabularul CONTEXT devine FORTH și intră în QUIT

WARNING (—adr) U
variabilă-utilizator care conține indicatorul modului de scriere a erorii

WHERE (n1 n2—) F
afișează locul erorii la încărcarea dintr-un ecran, parametrii sînt cei lăsați de ERROR

WHILE (f—) F, C, I
utilizat în bucla: **BEGIN** *listă1* **WHILE** *listă2* **REPEAT**
dacă f=1 continuă execuția buclei
dacă f=0 se iese imediat din buclă (execuția continuă cu primul cuvînt de după REPEAT)

WIDTH (—adr) U
variabilă-utilizator, conține lungimea maximă a unui cuvînt Forth (31 caractere)

WORD (c—) F
separă următorul șir de caractere delimitat de c din BLK și îl introduce în HERE

X1 (—adr)
variabilă în dublă lungime, utilizată de DRAW

XOR (n1 n2—n3) F
efectuează sau-exclusiv pe biți între n1 și n2, returnează rezultatul n3

Y1 (—adr)
variabilă în dublă lungime, utilizată de DRAW

[(—) F,C,I
trece temporar în mod execuție, cuvintele dintre [și] vor fi executate, nu compilate, se utilizează în pereche cu]

[COMPILE] *text* (—) F, C, I
compilează cuvîntul imediat text în dicționar, nu îl execută

] (—) F,C
trece înapoi în mod compilare, se utilizează în pereche cu [

6.6 Codurile erorilor

0 şirul nu poate fi identificat ca cuvînt (cuvînt inexistent în vocabularul **CONTEXT** sau ierarhic superioare)
1 stivă vidă
4 definiţie dublă
6 număr de bloc eronat
7 debordare stivă şi/sau dictionar prea mare (ocupă aceeaşi zonă de memorie)
9 ecranul 0 este considerat ca fiind atribuit consolei
17 cuvînt utilizat doar în mod compilare
18 cuvînt utilizat doar în mod execuţie
19 structuri de control **BEGIN, CASE, DO, IF** sau **REPEAT** incorect definite
20 definiţie terminată incorect
21 definiţie protejată la ştergere
22 **--->** utilizat numai în timpul încărcării **LOAD**
23 numărul de linie dintr-un ecran este maxim 15
24 vocabularul **CONTEXT** este diferit de vocabularul **CURRENT**

7 Prolog

7.1 Instalarea*

Prolog diferă în mod esenţial de toate celelalte limbaje de programare, cu excepţia Lisp, fiind un limbaj declarativ (nu imperativ). După încărcarea cu **LOAD ""** programul intră în execuţie în SUPERVIZOR, care este bucla sistemului de operare.
SUPERVIZOR-ul aşteaptă comenzi pe care le execută direct sau "date" de adăugat la baza de date. Toate comenzile necesită un argument (care pentru unele primitive poate fi orice) şi numai unul (prin folosirea anumitor primitive se poate forţa citirea în continuare a buffer-ului de intrare şi defini comenzi cu mai multe argumente).

Taste functionale:

Enter	redă controlul SUPERVIZOR-ului pentru interpretarea buffer-ului de intrare	
Break	(Symbol Shift & Space	întrerupe execuţia programelor
Cursor ->	(Caps Shift & 8)	folosite pentru editarea buffer-ului de intrare
Cursor <-	(Caps Shift & 5)	
SS&A	(Symbol Shift & A)	întrerupe temporar execuţia unei comenzi
Delete	(Caps Shift & 0)	şterge caracterul din stînga cursorului

Se pot utiliza cursoarele L, C, G şi E pentru accesarea literelor mari, mici, cifrelor şi simbolurilor grafice diverse.
Prompt-ul sistemului este compus din 2 părţi:
prompt-ul pus de SUPERVIZOR
- **&** pentru spaţiul de lucru
- *numele-modulului* pentru module
- *cifră* care reprezintă numărul de paranteze rotunde rămase deschise din buffer-ul de intrare introdus anterior

prompt-ul . pus de rutina de citire de la tastatură

*Micro-PROLOG este produs înregistrat al firmei Logic Programming Associates Ltd.

este reluat de fiecare dată cînd o comandă sau dată (de adăugat la baza de date) nu a fost terminată după Enter.

Buffer-ul de intrare nu este golit după citirea termenilor astfel încît pot fi introduse mai multe comenzi pe linie.

Pentru editarea programelor se utilizează primitivele ADDCL, CL, DELCL şi RFILL (sau editoare construite pe baza acestor primitive)

7.2 Structuri de limbaj

7.2.1 Termeni

Constante

Sînt şiruri de caractere de lungime maximă 60 caractere. Se face distincţia între literele mari şi mici. Sublinierea _ şi minus - sînt considerate litere. Cu excepţia literelor şi cifrelor celelalte caractere sînt considerate separatori. Dacă o constantă conţine separatori, atunci aceasta se introduce între ghilimele: "*constanta* "; utilizarea unei constante care poate fi confundată cu un număr impune utilizarea ghilimelelor. Ghilimelele introduse într-o constantă sînt precedate de @. Caracterele neafişabile (de control) sînt formate din @ urmat de un caracter al cărui cod ASCII este mai mare cu 64 decît codul caracterului neafişabil

Variabile

Literele **X, Y, Z, x, y, z** urmate (imediat) sau nu de un număr-index sînt variabile. În general variabilele pot înlocui orice: termeni, liste (întregi sau fragmente), nume de predicate, atomi, clauze (cu unele restricţii la unele predicate care acceptă ca variabile asignate termeni de un tip specificat). Numai ultimele 2 cifre ale numărului-index sînt semnificative. Într-o clauză pot exista maxim 64 de variabile. Variabilele sînt locale şi numele lor pot fi modificate de Prolog dar cu 2 restricţii: variabilele cu nume diferite rămîn cu nume diferite; numele este modificat la fel în toate poziţiile în care apare variabila

Numere

Întreg pozitiv: o secvenţă de cifre zecimale fără semnul +

Întreg negativ: o secvenţă de cifre zecimale precedată de semnul -

Număr pozitiv în virgulă flotantă: o secvenţă de cifre zecimale fără semnul + care conţine un punct zecimal . ; opţional poate fi urmat de un exponent care este un număr întreg precedat de **E** sau **e**. Punctul zecimal trebuie să fie precedat de cel puţin o cifră.

Număr negativ în virgulă flotantă: are acelaşi format ca numărul pozitiv în virgulă flotantă, fiind precedat de -

Numerele în virgulă flotantă pot fi introduse în orice formă, dar sînt afişate într-o formă standard (mantisa între -10 şi 10):

cifră . cifră ... cifră **E** *cifră cifră*

dacă exponentul este 0 atunci acesta nu se mai afişează.

Numerele întregi sînt în domeniul -32767, 32767.
Exponenţii trebuie să fie în domeniul -127, 127.

7.2.2 Liste

Lista reprezintă un şir de termeni încadraţi de paranteze rotunde:
(termen ... termen)
Listele pot fi imbricate una în alta nerestrictiv.
Are sens lista vidă: ( )
Caracterul | separă capul listei de o lungime şi/sau structură determinată de coada listei care poate avea o lungime/structură variabilă. Capul listei trebuie să conţină cel puţin un termen (eventual lista vidă).

7.2.3 Atomi

Atomul este o listă de termeni din care primul este numele unui predicat logic, iar următorii sînt argumentele sale:
(nume-predicat termen-1 ... termen-n)
Predicatele reprezintă o relaţie logică între termenii săi.
Rezultatul evaluării unui atom este o valoare logică: adevarat sau fals.
Predicatele predefinite în Prolog se numesc primitive.
Pot exista predicate care nu au nici un argument sau cu număr variabil de argumente.

7.2.4 Clauze

Clauzele reprezintă liste de atomi cu ajutorul cărora se definesc predicatele. Primul atom din clauză cuprinde predicatul de definit. Un predicat poate fi definit prin mai multe clauze, ca alternative.

((nume-predicat-de-definit termen-1 ... termen-n)
(predicat termen ... termen)
...
(predicat termen ... termen))

La evaluare primul atom returnează adevărat dacă toţi ceilalţi returnează adevărat la evaluarea în ordine.
Dacă o clauză (la evaluare) returnează fals se încearcă evaluarea clauzelor următoare (dacă nu este ultima).
În baza de date nu contează ordinea clauzelor referitoare la predicate diferite, dar pentru un acelaşi predicat de definit clauzele sînt ordonate conform introducerii lor de la tastatură (ordinea contează pentru clauze definind acelaşi predicat).

Predicatele pot fi definite (în clauze) în mod recursiv nerestrictiv.
O clauză poate conţine numai atomul de definit:
(*predicat-de-definit termen-1 ... termen-n*))
caz în care acesta este întotdeauna adevărat.

7.2.5 Comenzi

Introducerea unei linii de tipul:
nume-de-predicat-unar argument
produce tentativa de validare (prin backtraking) a atomului:
(*nume-de-predicat-unar argument*)
unde *argumentul* poate fi un termen sau o listă.
Dacă atomul returnează adevărat se afişează prompt-ul (care semnalizează aşteptarea introducerii unei noi comenzi sau clauze).
Dacă atomul returnează fals se afişează ? apoi prompt-ul.

7.2.6 Module

Modulele sînt colecţii închise de definiţii (clauze) care nu pot fi modificate din exterior (spaţiul de lucru sau alte module) şi a căror utilizare (cu excepţia celor specificate explicit) este transparentă în exterior (modulele sînt analogul subrutinelor din alte limbaje de programare) constantele fiind locale modulului (cu excepţia celor specificate explicit).
Modulul este caracterizat de:
- numele modulului: constantă diferită de numele predicatelor utilizate şi/sau numele fişierelor utilizate
- lista-export: cuprinde numele predicatelor definite în modul; care pot fi utilizate în alte module şi/sau spaţiul de lucru
- lista-import: cuprinde numele predicatelor definite în alte module şi/sau spaţiul de lucru şi utilizate de clauzele din modul

Toate constantele nespecificate în cele 2 liste sînt locale şi pot fi utilizate cu semnificaţie diferită în alte module şi/sau spaţiul de lucru (fiecare modul posedă un dicţionar propriu).
Primitivele Prolog sînt definite într-un modul distinct, inaccesibil utilizatorului (definiţiile lor nu pot fi modificate) şi numele lor sînt exportate implicit în toate modulele şi în spaţiul de lucru.

7.3 Modul de lucru

Pentru validarea unui atom se încearcă toate variantele care ar putea fi soluţii, prin metoda backtraking (căutare înapoi prin reevaluare).
Atomul de validat este comparat cu clauzele de definire ale predicatului corespunzator, din baza de date, în ordine.
Variabilelor nu li se atribuie valori decît dacă este strict necesar, o variabilă putînd înlocui orice.
O variabilă neasignată se potriveşte cu alta variabilă neasignată, caz în care devin legate şi asignarea unei variabile implică asignarea tuturor variabilelor legate de ea.
În cazul în care pentru atomul de validat este găsită o clauză pentru care se potriveşte cu primul atom (din clauză) Prolog memorează un pointer către următoarea clauză (pentru predicatul respectiv) într-o stivă şi încearcă validarea atomilor care compun clauza. Dacă validarea este reuşită (succesiv, după acceaşi metodă) Prolog semnalizează acest lucru, terminînd programul (apare prompt-ul curent). Dacă validarea este nereuşită atunci se descarcă pointer-ul din stivă şi este încercată clauza următoare. Dacă validarea eşuează (atomul returnează fals) Prolog semnalizează acest lucru afişînd ? (urmat de prompt-ul curent).
Stiva nu se încarcă în 3 situaţii:
- clauza predicatului definit este ultima din baza de date
- clauza specifică în mod explicit că este ultima/singura care poate fi încercată pentru validarea atomului
- atomul specifică în mod explicit (sau implicit pentru unele primitive) că admite soluţie unică

7.4 Primitive Prolog

ABORT

ABORT		(atom)
ABORT *termen*		(comandă)

Produce abandonarea tuturor evaluărilor (resetează stiva); *termen* poate fi orice.

ADDCL

(**ADDCL** *clauză nr*)	(a)	(atom)
(**ADDCL** *clauză*)	(b)	(atom)
ADDCL *clauză*	(c)	(comandă)

unde:

(a) adaugă clauza în baza de date după clauza cu numărul de ordine *nr*
(b),(c) adaugă clauza la sfîrşitul bazei de date; returnează adevărat; *clauza*, *nr* pot fi înlocuite cu variabile care trebuie să fie asignate în

momentul evaluării

ALL

ALL

Nume pentru baza de date din spaţiul de lucru/modulul curent, utilizat de LIST şi KILL.

BORDER

(BORDER *nr* **)** (atom)
BORDER *nr* (comandă)

Setează culoarea BORDER conform codului culorii nr:

0 negru	4 verde
1 albastru	5 cyan
2 roşu	6 galben
3 magenta	7 alb

Returnează adevărat, numărul este prelucrat modulo 8 şi poate fi înlocuit cu o variabilă care trebuie să fie asignată în momentul evaluării.

BP

(BP *nr1 nr2* **)** (atom)

Emite un semnal sonor la difuzor de frecvenţă f (în Hz) şi durată t (în s) unde:
nr1 =f*t; *nr2* =3.5E6/f/8-30.125

do	261.63Hz	fa#	369.99Hz
do#	277.18Hz	sol	392Hz
re	293.66Hz	sol#	415.30Hz
re#	311.13Hz	la	444Hz
mi	329.63Hz	la#	466.16Hz
fa	349.23Hz	si	493.88Hz

La fiecare salt la octava superioară/inferioară frecvenţele se dublează/înjumătăţesc.
Returnează adevărat, numerele pot fi înlocuite cu variabile care trebuie să fie asignate în momentul evaluării.

CHAROF

(CHAROF *caracter nr* **)** (atom)

Returnează adevărat dacă caracter are codul ASCII nr, termenii pot fi înlocuiţi cu variabile din care cel puţin una, oricare, trebuie să fie asignată în momentul evaluării, cealaltă, dacă este neasignată, capătă valoarea corespunzătoare.

CL

(CL *clauză* **)** (a) (atom)
(CL *clauză nr1 nr2* **)** (b) (atom)
CL *clauză* (c) (comandă)

Returnează adevărat dacă:
(a), (c) clauza există în baza de date
(b) clauza există în baza de date şi are numărul de ordine *nr2*, *nr2* > = *nr1*, *nr1* numărul de ordine de unde începe căutarea
Predicatul care se defineşte în clauză şi *nr1* trebuie să fie cunoscute în momentul evaluării (pot fi variabile dar asignate în momentul evaluării).

CLMOD

CLMOD (atom)
CLMOD *termen* (comandă)

Închide modulul curent revenind în spaţiul de lucru, termen poate fi orice; returnează adevărat; spaţiul de lucru nu poate fi închis.

CLOSE

(CLOSE *nume-fişier* **)** (atom)
CLOSE *nume-fişier* (comandă)

Închide fişierul *nume-fişier* deschis cu **OPEN** sau **CREATE**; se utilizează **CLOSE ""** dacă s-a utilizat **OPEN ""**. Returnează adevărat dacă *nume-fişier* coincide cu numele fişierului deschis şi poate fi închis.

CLS

(CLS *nr* **)** (atom)
CLS *nr* (comandă)

unde *nr* este număr prelucrat modulo 32 si semnifică:

cod-culoare-PAPER **+8*** *cod-BRIGHT* **+16*** *cod-FLASH*

unde:

cod-culoare-PAPER		*cod-BRIGHT*	*cod-FLASH*
0 negru	4 verde	0 normal	0 normal
1 albastru	5 cyan	1 strălucitor	1 strălucitor
2 roşu	6 galben		
3 magenta	7 alb		

CLS şterge ecranul şi setează culorile conform *nr*. Returnează adevărat, numărul poate fi înlocuit cu o variabilă care trebuie să fie asignată în momentul evaluării.

CMOD

(CMOD X) (atom)
CMOD X (comandă)

Returnează adevărat dacă X este variabilă neasignată şi simplă (nestructurată). Variabila este asignată cu numele modulului curent (& pentru spaţiul de lucru).

CON

(CON *termen* **)** (atom)
CON *termen* (comandă)

Returnează adevărat dacă termen este o constantă sau o variabilă asignată cu o constantă în momentul evaluării.

"CON:"

"CON:"

Numele fişierului consolă deschis permanent pentru citit (tastatură) şi scris (ecran). Tentativele CREATE, OPEN şi CLOSE pentru acest fişier nu generează fals sau eroare dar nu au nici o acţiune.

CREATE

(CREATE *nume-fişier* **)** (atom)
CREATE *nume-fişier* (comandă)

Creează un fişier pe bandă magnetică deschis pentru scris. Returnează adevărat dacă nu există alte fişiere deschise la momentul evaluării, din cele care pot fi închise. Numele fişierului este o constantă din 1-8 caractere. Scrierea efectivă (salvarea pe bandă) are loc în momentul închiderii fişierului.

CRMOD

(CRMOD *nume-modul listă-export listă-import* **)** (atom)

Creează un modul numit nume-modul unde:

nume-modul trebuie să fie o constantă (variabilă asignată) diferit de numele altor module, predicate definite şi constante utilizate în modulul respectiv sau în spaţiul de lucru

lista-export cuprinde numele predicatelor exportate de modul, acestea vor fi definite ulterior în cadrul modulului

lista-import cuprinde numele predicatelor importate în modul din spaţiul de lucru sau alte module; predicatele importate pot fi listate dar nu pot fi şterse

decît în modulul/spaţiul de lucru unde au fost definite; predicatele importate trebuie să fie definite; primitivele Prolog sînt implicit exportate în toate modulele şi în spaţiul de lucru

Returnează adevărat dacă condiţiile de creeare a modulului sînt respectate. Utilizatorul este lăsat în modulul respectiv după creearea acestuia dacă nu mai urmează nici un atom/clauză de evaluat.

DELCL

(DELCL *clauză* **)**	(a)(atom)
DELCL *clauză*	(b)(comandă)
(DELCL *predicat nr* **)**	(c)(atom)

Şterge clauza respectivă din baza de date:

(a),(b) şterge *clauza* , aceasta poate conţine variabile neasignate dar trebuie să fie în mod univoc determinată

(c) şterge clauza cu numărul de ordine *nr* referitoare la *predicat* (*predicat* şi *nr* pot fi variabile dar asignate)

Returnează adevărat dacă condiţiile de ştergere a clauzei sînt îndeplinite.

DICT

(DICT *nume-modul listă-export listă-import listă-constante* **)** (atom)

Reprezintă atomul de stare al spaţiului de lucru/modulului curent

Pentru module:

nume-modul	numele modulului
lista-export	lista predicatelor exportate de modul
lista-import	lista predicatelor importate de modul
lista-constante	lista predicatelor definite în modul

Pentru spaţiul de lucru:

nume-modul	&
lista-export	( ), spaţiul de lucru nu exportă nimic
lista-import	lista predicatelor exportate de toate celelalte module
lista-constante	lista predicatelor definite în spaţiul de lucru şi predicatul "?ERROR?"de mascare a erorilor

EQ

(EQ *termen1 termen2* **)** (atom)

Returnează adevărat dacă *termen1* este identic cu *termen2* (listele sau variabilele au aceeaşi structură). Variabilele devin asignate sau legate.

FAIL

FAIL	(atom)
FAIL *orice*	(comanda)

Returnează întotdeauna fals.

FORALL

(FORALL *termen1 termen2* **)** (atom)

unde *termen1* şi *termen2* sînt liste de atomi.

Returnează adevărat dacă pentru toate variabilele locale pentru care *termen1* este adevărat atunci şi *termen2* este adevărat. Variabilele globale din termeni trebuie să fie asignate în momentul evaluării altfel se pot genera răspunsuri greşite (Prolog nu verifică îndeplinirea acestei condiţii).

HYBRID

HYBRID	(atom)
HYBRID *orice*	(comandă)

Comută zona de dialog a ecranului pe ultimile 4 linii şi şterge ecranul conform ultimei specificaţii CLS (eliberează zona superioară a ecranului); avînd acţiune inversă lui NORMAL. Returnează adevărat.

IF

(IF *termen1 termen2 termen3* **)** (atom)

unde *termen1/ 2/ 3* sînt liste de atomi. Returnează adevărat dacă *termen1* şi *termen2* sînt adevăraţi sau *termen1* fals şi *termen3* adevărat.

INKEY

(INKEY X) (atom)
INKEY X (comanda)

Asignează **X** cu codul tastei apăsate, comută cursorul în L înainte de citire, caracterele neliterare sînt încadrate în ghilimele, cele neafişabile sînt precedate de @ urmat de un caracter al cărui cod ASCII este cu 64 mai mare decît al caracterului neafişabil. Nu aşteaptă apăsarea unei taste, în cazul în care nu a fost apăsată nici o tastă **X** este asignat cu un şir vid "". **X** trebuie să fie variabilă neasignată în momentul evaluării. Returnează adevărat.

INT

(INT *nr* **)** (atom)
(INT *nr* **X)** (atom)
INT *nr* (comanda)

unde:

(a),(c) returnează adevărat dacă *nr* este un număr întreg în intervalul -32767, +32767 (*nr* poate fi o variabilă, asignată în momentul evaluării)

(b) returnează adevărat dacă *nr* este în intervalul -32767, +32767 şi variabilei **X**, neasignate în momentul evaluării, i se atribuie partea întreagă a *nr*

INTOK

(INTOK *nume-fişier* **X)** (atom)

Citeşte un termen din fişierul *nume-fişier* care trebuie să fie deschis pentru citit şi îl atribuie variabilei neasignate **X**. Termenul trebuie să fie o constantă sau un număr, inserează ghilimele pentru constantele care conţin separatori sau pot fi interpretate ca variabile. Returnează adevărat.

ISALL

(ISALL *termen1 termen2 termen3* **)** (atom)

Returnează adevărat dacă *termen1* este lista tuturor soluţiilor posibile de tipul *termen2* care reprezintă o variabilă locală sau o listă de variabile locale şi care verifică *termen3* care este o listă de atomi. Variabilele globale din termeni trebuie să fie asignate în momentul evaluării altfel se pot genera răspunsuri greşite (Prolog nu verifică îndeplinirea acestei condiţii). *Termen1* reprezintă lista soluţiilor în ordinea inversă găsirii lor.

KILL

(KILL *nume* **)** (atom)
KILL *nume* (comandă)

Dacă *nume* este un nume de predicat atunci şterge toate clauzele de definiţie din toate modulele sau spaţiul de lucru referitoare la predicat. Dacă *nume* este nume de modul atunci şterge modulul. Dacă *nume* este **ALL** şterge toate clauzele din spaţiul de lucru/modulul curent. Un modul nu poate fi şters din interiorul său. *Nume* poate fi o variabilă asignată în momentul evaluări. Returnează adevărat.

LESS

(LESS *termen1 termen2* **)** (atom)

Returnează adevărat dacă *termen1* este strict mai mic decît *termen2* conform ordinii lexicografice pentru constante (numerele sînt mai mici decît literele majuscule care sînt mai mici decît literele minuscule) sau ordinii normale pentru numere (constantele care pot fi confundate cu numere trebuie încadrate în ghilimele. Variabilele trebuie asignate la momentul evaluării.

LIST

(LIST *nume* **)** (atom)
LIST *nume* (comandă)

Scrie la consolă toate clauzele de definiţie (eventual nici una) referitoare la predicatul *nume*, definite în spaţiul de lucru/modulul curent sau importate de acesta. Dacă *nume* este **ALL** listează toate clauzele existente în spaţiul de lucru/modulul curent. *Nume* poate fi înlocuit cu o variabilă asignată în momentul evaluării. Returnează adevărat cu excepţia cazului în care *nume* este numele unei primitive Prolog care nu poate fi listată.

LISTP

(LISTP *nume-fişier nume* **)** (atom)
LISTP *nume* (comandă)

Scrie în fişierul *nume-fişier* toate clauzele de definiţie (eventual nici una) referitoare la predicatul *nume*, definite în spaţiul de lucru/modulul curent sau importate de acesta. Dacă *nume* lipseşte (este opţional) se scriu toate clauzele existente în spaţiul de lucru/modulul curent. Nume poate fi înlocuit cu o variabilă asignată în momentul evaluării. Returnează adevărat cu excepţia cazului în care nume este numele unei primitive Prolog care nu poate fi listată.

LNE

(LNE *nr1 nr2 nr3 nr4 nr5 nr6* **)** (atom)

Trasează o linie între punctele de coordonate (*nr1*, *nr2*) şi (*nr3*, *nr4*)unde:

nr1, *nr3* între -128 şi +127, de la stînga la dreapta
nr2, *nr4* între -88 şi +87, de jos în sus
nr5 este opţional şi are valoarea (modulo 256):
cod-culoare-INK **+8***_cod-culoare-PAPER_
+64*_cod-BRIGHT_ **+128***_cod-FLASH_
valoare implicită: cea anterioară
nr6 este opţional şi are valoarea (modulo 2):
0 suprascriere (OVER); 1 normal
valoare implicită: 0

cod-culoare:		*cod-BRIGHT*	*cod-FLASH*
0 negru	4 verde	0 normal	0 normal
1 albastru	5 cyan	1 strălucitor	1 strălucitor
2 roşu	6 galben		
3 magenta	7 alb		

Returnează adevărat dacă *nr5*, *nr6* sînt întregi între -32767 şi 32767 (*nr1*/2/3/4 sînt rotunjiţi la partea întreagă); dacă *nr1* =*nr3* şi *nr2* =*nr4* nu se trasează nimic).

LOAD

(LOAD *nume* **)** (atom)
LOAD *nume* (comandă)

Citeşte un fişier *nume* de pe bandă magnetică, dacă *nume* este "" se citeşte primul fişier întîlnit. Fişierele sînt formate dintr-un număr de blocuri de lungime 260 bytes fiecare, dacă unul din blocuri s-a încărcat cu eroare se reia citirea de la acest bloc. Returnează adevărat.

LST

(LST *termen* **)** (atom)
LST *termen* (comandă)

Returnează adevărat dacă *termen* este de tip listă.

"LST:"

"LST:"

nume de fişier-imprimantă deschis pentru scris.

NEW

NEW (atom)
NEW *orice* (comandă)

Reiniţializează total sistemul Prolog dar nu afectează culorile CLS şi modul NORMAL/HYBRID anterioare.

NORMAL

NORMAL (atom)
NORMAL *orice* (comandă)

Comută zona de dialog a ecranului pe primele 22 de linii şi şterge ecranul conform ultimei specificaţii **CLS**, avînd acţiune inversă lui HYBRYD. Returnează adevărat (iniţial NORMAL).

NOT

(NOT *predicat termen-1 ... termen-n* **)** (atom)
NOT *comandă* (comandă)

Dacă atomul (*predicat termen-1 ... termen-n*) este adevărat atunci atomul **(NOT** *predicat termen-1 ... termen-n* **)** returnează fals şi reciproc. În regim de comandă **NOT** anulează comanda şi returnează adevărat.

NOT poate fi utilizat doar pentru verificare, variabilele globale trebuie să fie asignate în momentul evaluării. Înaintea evaluării **NOT**, este evaluat atomul care conţine predicatul astfel încît dacă această evaluare implică o acţiune, ea va fi executată.

NUM

(NUM *termen* **)** (atom)
NUM *termen* (comandă)

Returnează adevărat dacă *termen* este un număr (sau o variabilă asignată cu un număr).

OPEN

(OPEN *nume-fişier* **)** (atom)
OPEN *nume-fişier* (comandă)

Deschide un fişier pe bandă magnetică pentru citit. **OPEN ""** va citi primul fişier întîlnit. Returnează adevărat.

OPMOD

(OPMOD *nume-modul* **)** (atom)
OPMOD *nume-modul* (comandă)

Deschide modulul *nume-modul*. Returnează adevărat dacă modulul există şi poate fi deschis (un modul nu poate fi deschis din el însuşi). În mod comandă utilizatorul este lăsat în modulul deschis. În mod evaluare de clauză execuţia continuă cu evaluarea atomului următor.

OR

(OR *listă-atomi-1 listă-atomi-2* **)** (atom)

Returnează adevărat dacă *lista-atomi-1* sau *lista-atomi-2* (sau amîndouă) sînt adevărate.

P

(**P** *termen-1 ... termen-n*) (atom)
P *termen* (comandă)

Scrie seria de termeni *termen-1 ... termen-n* la consolă, nu efectuează retur de car după scriere. Pentru variabilele asignate scrie valorile corespunzătoare, numele variabilelor neasignate pot fi modificate. Returnează adevărat.

PIO

(**PIO** *nr termen*) (atom)

Citeşte sau scrie la portul *nr* (poate fi variabilă asignată), între -32767 şi +32767 (întreg). *Nr* este convertit intern în număr fără semn, între 0 şi 65535 (complement faţă de 2). Dacă *termen* este număr cunoscut (variabilă asignată) între -32767 şi +32767 acesta este convertit intern în număr fără semn, între 0 şi 65535 (complement faţă de 2)şi byte-ul mai puţin semnificativ este trimis la portul specificat (se execută instrucţiunile **LD BC,** *nr* ; **LD A,** *termen* ; **OUT (C), A**). Dacă *termen* este variabilă neasignată citeşte de la port o valoare şi atribuie variabilei numărul citit, între 0 şi 255 (se execută instrucţiunile **LD BC,** *nr* ; **IN (C), A**; **LD** *termen* **, A**). Returnează adevărat dacă valorile date sînt în domeniul specificat şi întregi.

PNT

(**PNT** *nr1 nr2 nr3 nr4*) (atom)

Desenează un punct de coordonate (*nr1, nr2*) unde:

nr1 între -128 şi +127, de la stînga la dreapta
nr2 între -88 şi +87, de jos în sus
nr3 este opţional şi are valoarea (modulo 256):
cod-culoare-INK **+8***_cod-culoare-PAPER_
+64*_cod-BRIGHT_ **+128***_cod-FLASH_
valoare implicită: cea anterioară
nr4 este opţional şi are valoarea (modulo 2):
0 suprascriere (OVER); 1 normal
valoare implicită:0

cod-culoare:		*cod-BRIGHT*	*cod-FLASH*
0 negru	4 verde	0 normal	0 normal
1 albastru	5 cyan	1 strălucitor	1 strălucitor
2 roşu	6 galben		
3 magenta	7 alb		

Returnează adevărat dacă *nr3, nr4* sînt întregi între -32767 şi 32767 (*nr1/2* sînt rotunjiţi la partea întreagă).

PP

(**PP** *termen-1 ... termen-n*) (atom)
PP *termen* (comanda)

Scrie seria de termeni *termen-1 ... termen-n* la consolă, efectuează retur de car după scriere. Pentru variabilele asignate scrie valorile corespunzătoare, numele variabilelor neasignate pot fi modificate. Returnează adevărat.

R

(**R X**) (atom)
R X (comandă)

Citeşte termenul următor din buffer-ul de intrare (consolă) dacă acesta nu a fost citit complet sau de la tastatură şi îl atribuie variabilei neasignate **X** în momentul evaluării (termenii pot fi orice). Returnează adevărat.

READ

(**READ** *nume-fişier* **X**) (atom)
READ *nume-fişier* **X** (comandă)

Citeşte termenul următor din fişierul *nume-fişier* şi îl atribuie variabilei neasignate **X** în momentul evaluării (termenii pot fi orice). Returnează adevărat.

RFILL

(RFILL (*termen-1 ... termen-n*) **X**) (atom)

Introduce în editare (pe linie, cu ajutorul cursorului) lista de termeni. Variabila **X** neasignată la momentul evaluării capătă valoarea listei de termeni (modificată sau nu de utilizator) după apăsarea tastei Enter. Returnează adevărat.

RND

(RND X *nr*) (atom)

Numărul întreg *nr* între -32767 şi 32767 (variabila asignată) este convertit intern într-un număr întreg fără semn, între 0 şi 65535 (complemant faţă de 2) şi apoi se calculează un număr aleator între 0 şi *nr* -1 cu care este asignată variabila **X** (la afişare acesta va fi considerat ca număr cu semn, în complement faţă de 2, între -32767 şi +3276). Returnează adevărat dacă X este variabilă neasignată la momentul evaluării.

SAVE

(SAVE *nume-fişier nume-predicat*) (a)(atom)
(SAVE *nume-fişier*) (b)(atom)
SAVE *nume-fişier* (c)(comandă)

Scrie într-un fişier pe bandă cu numele *nume-fişier* :
(a) toate clauzele referitoare la *nume-predicat*
(b),(c) toate clauzele
Returnează adevărat.

SIGN

(SIGN *nr* **X**) (atom)

Atribuie variabilei neasignate **X** valoarea 1 dacă *nr* este număr pozitiv sau valoarea **-1** dacă *nr* este număr negativ (*nr* poate fi variabilă asignată) Returnează adevărat.

SPACE

(SPACE X) (atom)

Atribuie variabilei neasignate **X** numărul de kbytes care mai sînt disponibili în memorie pentru program (iniţial 19). Returnează adevărat.

STRINGOF

(STRINGOF *listă constantă*) (atom)

Returnează adevărat dacă *listă* este lista tuturor caracterelor, în aceeaşi ordine, din care este formată *constanta*; *lista* şi *constanta* pot fi variabile dintre care cel puţin una (oricare) trebuie să fie asignată în momentul evaluării.

SUM

(SUM *nr1 nr2 nr3*) (atom)

Returnează adevărat dacă numerele *nr1/2/3* sînt în relaţia: *nr1 +nr2 =nr3. nr1/2/3* pot fi variabile din care cel puţin 2 (oricare) trebuie să fie asignate în momentul evaluării.

SYS

(SYS *constantă*) (atom)
SYS *constantă* (comandă)

Returnează adevărat dacă *constanta* (variabilă asignată) este numele unei primitive Prolog.

TIMES

(TIMES *nr1 nr2 nr3*) (atom)

Returnează adevărat dacă numerele *nr1/2/3* sînt în relaţia:*nr1 * nr2 = nr3. nr1/2/3* pot fi variabile din care cel puţin 2 (oricare) trebuie să fie asignate în momentul evaluării.

VAR

(VAR X) (atom)
VAR X (comandă)

Returnează adevărat dacă **X** este o variabilă neasignată în momentul evaluării.

W

(W *nume-fişier listă-termeni*) (atom)

Scrie în fişierul *nume-fişier*, deschis pentru scris, termenii din listă, nu inserează retur de car. Returnează adevărat (cu excepţia cazului în care *listă-termeni* nu este listă).

WRITE

(WRITE *nume-fişier listă-termeni*) (atom)

Scrie în fişierul *nume-fişier*, deschis pentru scris, termenii din listă, inserează retur de car. Returnează adevărat (cu excepţia cazului în care *listă-termeni* nu este listă).

?

(? *listă-atomi*) (atom)
? *listă-atomi* (comandă)

Evaluează atomii din listă şi returnează adevărat dacă toţi atomii returnează adevărat (cu excepţia cazului în care *listă-atomi* nu este listă de atomi)

!

(! *nume-predicat termen-1 ... termen-n*) (atom)

Atenţionează Prolog că la evaluare atomul:

(nume-predicat termen-1 ...termen-n)

are soluţie unică astfel încît stiva nu se mai încarcă pentru backtraking. Nu se poate utiliza în atomul de definit.

/

/ (atom)

Atenţionează Prolog că clauza în care apare atomul / este ultima care poate fi folosită pentru găsirea soluţiilor, astfel încît stiva nu se mai încarcă pentru backtraking. Returnează adevărat.

/*

(*/** comentariu*) (atom)
/ comentariu* (comandă)

Permite introducerea comentariilor în programe. Returnează adevărat.

"< >"

(*"< >" termen*) (atom)
"< >" termen (comandă)

Reprezintă primitiva Prolog de interpretare a textului introdus la tastatură. Dacă *termen* este o comandă cu un singur parametru citeşte parametrul ei şi o execută; dacă *termen* este o clauză o adaugă la baza de date. Returnează adevărat dacă termen este o comandă (executată corect) sau o clauză.

"<SUP>"

("<SUP>") (atom)

Reprezintă programul SUPERVIZOR (bucla sistemului de operare).

"?ERROR?"

(("?ERROR?" *nr* **X)** *atom ... atom*) (clauză)

"?ERROR?" permite mascarea erorilor de tipul *nr* (dacă există clauza de mascare), asignează variabila **X** cu atomul care a generat eroarea şi continuă evaluarea cu lista de atomi din clauza de mascare a erorilor. Dacă toţi aceştia returnează adevărat, evaluarea continuă cu atomul de după cel care a generat eroarea.

7.5 Codurile erorilor

0 subdepăşire aritmetică (număr prea mic)
1 supradepăşire aritmetică (număr prea mare)
2 predicat nedefinit
3 prea multe sau prea puţine variabile
4 definiţie protejată
5 eroare de fişier
6 fişier rămas neînchis
9 eroare de scriere
11 Break
12 utilizare incorectă a modulelor
13 linie sau punct în afara ecranului
15 Break în timpul scrierii/citirii pe bandă magnetică
22 culoare incorectă

Sumar

1 RUTINELE ROM 3
1.1 Rutinele restart 3
1.2 Rutinele de tastatură 4
1.3 Rutine difuzor 5
1.4 Rutine casetofon 6
1.5 Rutine ecran 7
1.6 Rutine de execuţie 10
1.7 Interpretorul BASIC 14
1.8 Evaluatorul de expresii 18
1.9 Rutine aritmetice 21
1.10 Calculatorul în virgulă flotantă 22
1.11 Variabilele de sistem 28
1.12 Harta memoriei totale 31
1.13 Structuri BASIC 32
1.14 Setul de caractere 33

2 ASAMBLORUL/DEZASAMBLORUL 34
2.1 Microprocesorul Z80 34
2.2 Încărcare pe 8 biţi 35
2.3 Încărcare pe 16 biţi 36
2.4 Schimb, transfer de blocuri si căutare 38
2.5 Aritmetică şi logică pe 8 biţi 39
2.6 Control 41
2.7 Aritmetică pe 16 biţi 42
2.8 Rotaţie şi shift 43
2.9 Set, reset şi test pe biţi 45
2.10 Transfer 46
2.11 Subrutine 47
2.12 Input şi output 48
2.13 Tabelele de dezasamblare 50
2.13.1 Tabela 1 (principală), fără prefix 50
2.13.2 Tabela2; prefix: CB 52
2.13.3 Tabela 3; prefix ED 54
2.13.4 Tabela 4; prefix DD/FD pentru registrul IX/IY 55
2.13.5 Tabele de conversie: binar/zecimal/hexazecimal 56
2.14 Asamblorul GENS 57
2.14.1 Instalarea 57
2.14.2 Editorul 57
2.14.3 Directive de asamblare 61
2.14.4 Codul erorilor generate 63
2.15 Dezasamblorul MONS 63
2.15.1 Instalarea 63
2.15.2 Comenzi 64

3 BASIC 68
3.1 BASIC Spectrum 68

3.1.1 Constante, variabile, expresii 68
3.1.2 Cuvinte cheie BASIC Spectrum 69
3.1.3 Structura programului BASIC 83
3.1.4 Mesaje de eroare 83
3.2 Compilatorul HIBASIC. 85
3.2.1 Instalarea 85
3.2.2 Directive 85
3.3 Fifth 86
3.3.1 Instalarea 86
3.3.2 Erori 86
3.3.3 Comenzi 87
3.3.4 Funcţii 92
3.4 Beta BASIC 93
3.4.1 Instalarea 93
3.4.2 Comenzi 94
3.4.3 Funcţii 101
3.4.4 Mesaje de eroare 104

4 PASCAL 106
4.1 Instalarea 106
4.2 Editorul 107
4.2.1 Generarea textului sursă 107
4.2.2 Listare 107
4.2.3 Editare 107
4.2.4 Comenzi de bandă magnetică 108
4.2.5 Comenzi de uz general 109
4.2.6 Compilarea si rularea programelor 109
4.3 Structuri de limbaj 109
4.4 Manipularea datelor în memorie 115
4.5 Funcţii şi proceduri predefinite 116
4.5.1 Funcţii 116
4.5.2 Proceduri 118
4.6 Directive de compilare 119
4.7 Codurile erorilor 120
4.7.1 Erori la compilare 121
4.7.2 Erori la execuţie 121

5 C 123
5.1 Instalarea 123
5.2 Editorul 124
5.2.1 Generarea textului sursă 124
5.2.2 Listare 124
5.2.3 Editare 124
5.2.4 Comenzi de bandă magnetică 125
5.2.5 Comenzi de uz general 126
5.3 Structuri de limbaj 126
5.3.1 Introducere 126
5.3.2 Convenţii lexicale 126
5.3.3 Interpretarea identificatorilor 127
5.3.4 Obiecte şi s-valori (l-values) 128
5.3.5 Conversii 128
5.3.6 Expresii 129

5.3.7 Declaraţii 132
5.3.8 Instrucţiuni 134
5.3.9 Definiţii externe 136
5.3.10 Domeniile de valabilitate ale declaraţiilor . . . 136
5.3.11 Preprocesorul 136
5.3.12 Expresii constante 137
5.4 Funcţii predefinite 138
5.5 Codurile erorilor 139
5.6 Manipularea datelor în memorie 141
5.6.1 Dimensiunea tipurilor 141
5.6.2 Clase de memorie 141
5.6.3 Funcţii 141
5.6.4 Fişiere 142
5.6.5 Regiştrii Z80 142

6 FORTH 143
6.1 Instalarea 143
6.2 Structuri de limbaj 143
6.3 Structura memoriei 144
6.3.1 Harta memoriei 144
6.3.2 Structura cuvintelor 145
6.4 Editorul 146
6.5 Cuvinte predefinite 147
6.6 Codurile erorilor 163

7 PROLOG 164
7.1 Instalarea 164
7.2 Structuri de limbaj 164
7.2.1 Termeni 165
7.2.2 Liste 165
7.2.3 Atomi 166
7.2.4 Clauze 166
7.2.5 Comenzi 167
7.2.6 Module 167
7.3 Modul de lucru 168
7.4 Primitive Prolog. 168
7.5 Codurile erorilor 178

BIBLIOGRAFIE

1.*** — **ZX Spectrum +— User Guide**
Sinclair Research Ltd. — London 1984

2.**Dr. Ian Logan, Dr. Frank O'Hara**
The Complete Spectrum ROM Dissasembly
Melbourne House Publishers — Nashville 1983

3.**Jean-Francois Sehan**
Clefs pour le ZX Spectrum et Timex 2000
Editions du P.S.I. — Torcy Marne-la-Valle 1983

4.**M. Patrubany**
Totul despre microprocesorul Z80
Editura Tehnică — Bucureşti 1989

5.*** — **HISOFT DEVPAC 3T**
HISOFT — 1984

6.**Richard Taylor**
Fifth — Users Manual
Computer Rentals Ltd.

7.*** — **Beta Basic**
Betasoft — 1984

8.*** — **HISOFT — Pascal 4T**
HISOFT — 1982

9.**Brian W. Kenirghan, Dennis M. Ritchie**
The C Programming Language
Prentice - Hall Inc. — New Jersey 1978

10.*** — **HISOFT C for ZX Spectrum**
HISOFT — 1984

11.**Radu Berindeanu, Agota Matecovits**
Forth, concept informatic şi limbaj de programare
Editura Facla — Timişoara 1991

12. **K. L. Clark, J. R. Ennals, F. G. McCabe**
A Micro PROLOG Primer
L. P. A. — London 1981

În scopul de a răspunde cît mai bine dorințelor dumneavoastră vă rugăm să completați acest chestionar și să îl trimiteți pe adresa editurii: Editura APH, str. Cap. Preda nr. 12, sector 5, 76437 București 69.

De unde ați fost informat de această carte ?

- ☐ Librărie (magazin). Care? . . . . . .
- ☐ Mi-a spus un prieten
- ☐ Am citit într-un ziar, revistă. Care? . . . . . .
- ☐ Altele. Vă rugăm specificați. . . . . .

. . . . . .

Ce vîrstă aveți?

☐ 10 - 15 ☐ 16 - 19 ☐ 20 - 24 ☐ peste 24

Ce părere aveți despre această carte ?

- ☐ Bună
- ☐ Potrivită
- ☐ Proastă
- ☐ Scumpă
- ☐ Preț mediu
- ☐ Ieftină

Vă rugăm să specificați ce alte teme ați dori să trateze Seria Informatică.

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Ce alte cărți v-ar interesa ?

1)Tehnico - Științifice

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

2)Beletristică

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Ocupația . . . . . .
Localitatea . . . . . .
Județul . . . . . .

Vă mulțumim.

Lucrarea contine informatii deosebit de utile tuturor utilizatorilor si în special programatorilor de calculatoare ZX SPECTRUM, multe dintre acestea fiind pentru prima oara publicate în România.

Sînt prezentate informatii cu privire la rutinele ROM (adresele lor si modul de utilizare), modul de utilizare al compilatoarelor PASCAL, C, FORTH, PROLOG, BASIC, BETABASIC precum si limbajul de asamblare al microprocesorului Z80 cu modul de utilizare al asamblorului/dezasamblorului GENS/MONS.

Cartea este conceputa ca un ghid în care informatiile necesare sa poata fi usor gasite si utilizate.

În curs de aparitie:

M. M. POPOVICI

– BASIC pentru calcularoarele ZX SPECTRUM, HC, TIM-S, COBRA, CIP, JET ...

• • ***COLECTIE DE PROGRAME***

Lucrarea contine programe tehnico-stiintifice, de matematica, de interes general, programe de divertisment (jocuri), precum si prezentarea programelor BETA BASIC si HISOFT BASIC însotite de aplicatii.

– LIMBAJUL MASINA al calculatoarelor ZX SPECTRUM, HC, TIM-S, COBRA, CIP, JET ...

Este prima lucrare care trateaza în mod unitar folosirea limbajului de asamblare Z80 si este ilustrata cu peste 150 de rutine care realizeaza spectaculoase efecte vizuale, sonore, de scriere, de animatie, etc.

ISBN 973-95175-6-0

LEI: ~~450~~

580